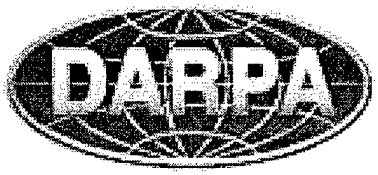
AFRL-IF-WP-TR-2001-1536

CHAMPION: A SOFTWARE DESIGN ENVIRONMENT FOR ADAPTIVE COMPUTING SYSTEMS AND APPLICATION SPECIFIC INTEGRATED CURCUITS (ASICs)



DR. DON BOULDIN

UNIVERSITY OF TENNESSEE
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
1508 MIDDLE DRIVE
KNOXVILLE, TX 37996-2100

JULY 2001

FINAL REPORT FOR PERIOD 29 SEPTEMBER 1997 – 30 APRIL 2001

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.

Al Scarpelli

AL SCARPELLI
Project Engineer/Team Leader
Embedded Info Sys Engineering Branch
Information Technology Division

James S. Williamson

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. AGENCY USE ONLY (leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Jul 2001	Final 29 September 1997 – 30 April 2001

4. TITLE AND SUBTITLE
CHAMPION: A Software Design Environment for Adaptive Computing Systems and Application Specific Integrated Circuits (ASICs)

5. FUNDING NUMBERS
C: F33615-97-C-1124
PE: 69199
PR: ARPA
TA: AS
WU: 08

6. AUTHOR(S)
Dr. Don Bouldin

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Tennessee, Department of Electrical and Computer Engineering
1508 Middle Drive
Knoxville TN 37996-2100

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 AFB OH 45433-7334
POC: Al Scarpelli, AFRL/IFTA, 937-255-6548 Ext. 3603

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-WP-TR-2001-1536

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
Adaptive Computing Systems (ACSs) can serve as flexible hardware accelerators for applications in domains such as image and signal processing. However, the mapping of applications onto ACSs using traditional methods can take months to develop and debug. To enable application designers to map their applications automatically, CHAMPION was developed to permit high-level design entry using the Khoros Cantata graphical programming environment and hide low-level details of the hardware architecture. The key idea underlying CHAMPION is its ability to reuse precompiled hardware modules written in VHDL. These modules produce identical results to fixed-point C modules installed in Cantata which the user interconnects graphically and simulates on a general purpose UNIX workstation. The resulting net-list is converted into a directed graph and manipulated by CHAMPION so that data widths and clock delays are matched. If the graph is too large for a single FPGA, then it is partitioned automatically. An automatic target recognition application was automatically mapped to a WildForce ACS containing five FPGAs, achieving a productivity gain of over 2000. Other moderately complex applications were mapped to multiple ACS platforms, as well as to single-chip ASICs. CHAMPION enables faster application development, ACSs to be utilized by a wider audience, and quick mapping onto multiple ACS platforms and ASICs, thereby exploiting rapid advances being made in hardware.

14. SUBJECT TERMS
adaptive computing systems; programmable logic; VHDL reuse;
designer productivity improvement

15. NUMBER OF PAGES
34

16. PRICE CODE

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

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Z39-18
298-102

THIS PAGE INTENTIONALLY LEFT BLANK

Table Of Contents

List of Figures

List of Tables

Executive Summary

Adaptive Computing Systems (ACSs) can serve as flexible hardware accelerators for applications in domains such as image and digital signal processing. However, the mapping of applications onto ACSs using the traditional methods can take months for a hardware engineer to develop and debug. To enable application designers to map their applications automatically onto ACSs, a software design environment called CHAMPION was developed at the University of Tennessee. This environment permits high-level design entry using the KHOROS/CANTATA graphical programming environment from KRI and hides from the user the low-level details of the hardware architecture.

The key idea underlying CHAMPION is its ability to reuse precompiled hardware modules written in VHDL. These modules produce identical results to fixed-point C modules installed in CANTATA which the user interconnects graphically and simulates on a general purpose UNIX workstation. The resulting net-list is converted into a directed graph and manipulated by CHAMPION so that data widths and clock delays are matched. If the graph is too large for a single FPGA, then it is partitioned automatically.

To benchmark CHAMPION, an automatic target recognition application containing 93 modules interconnected by 226 nets was captured using CANTATA. Mapping the net-list onto an Annapolis Micro Systems Wildforce ACS containing 5 FPGAs required 6 staff-weeks to map manually while CHAMPION was able to perform the mapping in less than 6 minutes. Thus, a productivity gain of over 2000 was demonstrated.

Additional validation of CHAMPION was performed using three other moderately complex applications. These were mapped to the Wildforce as well as the Wildcard and the USC-ISI SLAAC-1V ACS platforms. All of these applications were also mapped into single-chip Application Specific Integrated Circuits (ASICs) (0.5-micron CMOS).

Thus, CHAMPION enables application development to be accomplished in less time and ACSs to be utilized by a wider audience. Furthermore, CHAMPION provides the means to map onto multiple ACS platforms and ASICs, thereby exploiting rapid advances being made in hardware.

ACKNOWLEDGEMENT

The authors gratefully acknowledge the support of DARPA and the Air Force Research Laboratory under contract F33615-97-C-1124.

1. Introduction

Graphical programming environments such as KHOROS/CANTATA [1-2] from KRI, LabVIEW from National Instruments and Simulink from MathWorks, allow applications to be graphically represented as a set of functional blocks connected by signal paths as shown in Figure 1. By insulating the application programmer from low-level programming details, these environments allow faster and easier development of complex applications but the execution times on conventional CPUs are often long due to large input data or computationally intensive operators in the applications. For many types of commercial and military applications, which require high throughput, these long execution times are simply unacceptable.

With rapid advances in hardware, these complex applications can now be implemented in an Adaptive Computing System (ACS) composed of multiple Field-Programmable Gate Arrays (FPGAs) serving as general purpose processing elements as well as interfacing devices as depicted in Figure 2. Since FPGA-based computing systems can be tailored to the particular computational needs of a given application, they have been shown to have considerable performance advantages over conventional processor-based systems for certain types of applications [3-6].

Traditionally, the task of developing applications for an ACS requires considerable knowledge in digital hardware design and entails a long and tedious process, often requiring months to generate the Hardware Description Language (HDL) representation and then to synchronize, partition, and synthesize the digital circuit. Significant effort is also required to resolve the issue of the intricate interactions between the hardware (ACS) and the software (host machine). The lack of supportive design environments results in an unacceptably long turn-around time for leveraging the benefits of this type of hardware. Therefore, it is necessary to develop an end-to-end mapping tool that allows the designers to reduce the time required to move from specification to hardware implementation.

This objective has been achieved by the CHAMPION software design environment which provides automatic mapping of applications in the CANTATA graphical programming environment to ACSs. CHAMPION is a complete design environment that provides the tools needed to capture, simulate, and implement software applications on multiple ACSs. In this document, we present the design flow of this end-to-end design environment. The strength of this ACS-dedicated design flow includes its capability of yielding digital systems with high clock rates in low mapping time. It also allows CANTATA applications to be mapped onto multiple ACS hardware architectures such as the Wildforce board and Wildcard developed by AMS [3], and the SLAAC board [6] developed by the University of Southern California. Another advantage is that the design flow allows graphical programming environments other than CANTATA to be easily adapted as the design entry for CHAMPION.

The approach taken by CHAMPION is similar to that of several other research programs at Colorado State University and Northwestern University [7-8]. However, in our case, we perform synthesis and place/route on our library cells in advance. Thus, we have accurate information on the size and delay of each cell and only have to re-synthesize small netlists that represent the collection of cells that fit in each FPGA. The competing approaches merge the VHDL code into a single, large file that must be

fully re-synthesized and then partitioned at a finer grain than our approach. Hence, CHAMPION is presented with a netlist that is 10-100 times smaller and can be expected to execute in 100-1000 times less time while producing performance results within a few percent of the others.

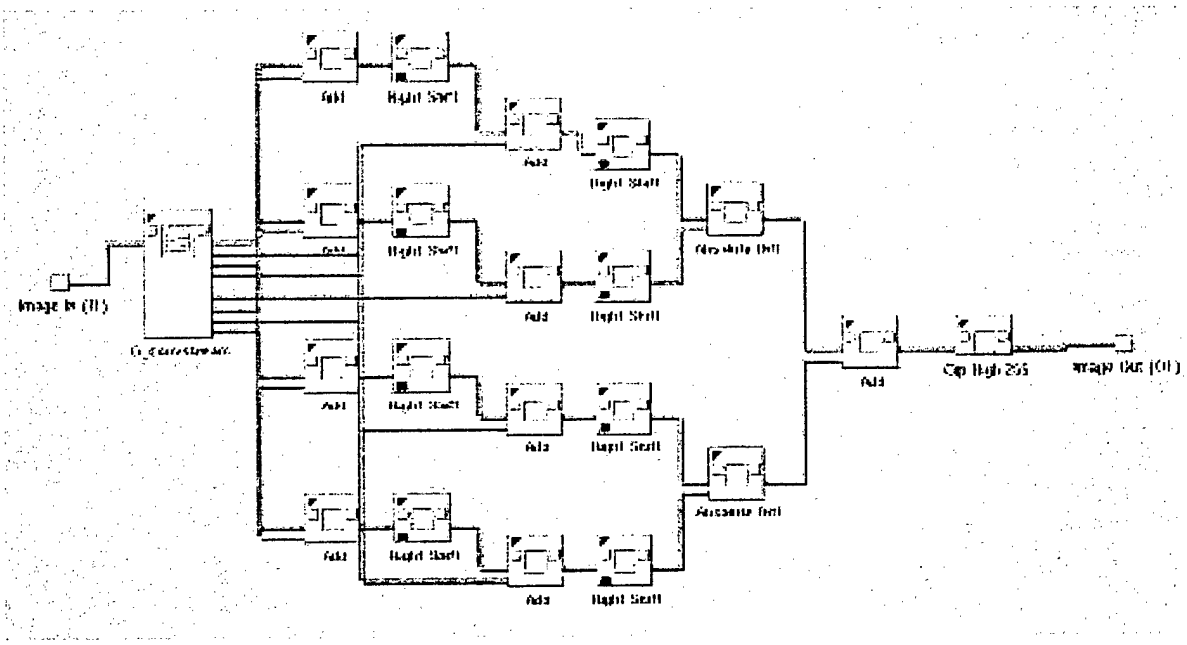


Figure 1. A KHOROS/CANTATA Workspace Describing the Design Flow of an Application

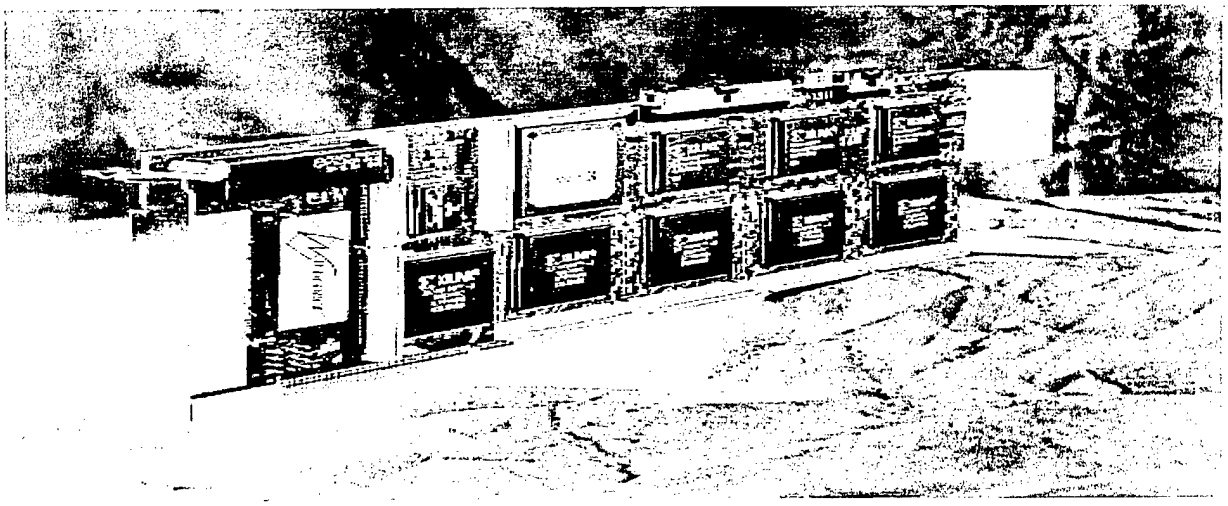


Figure 2. An Adaptive Computing System Containing Multiple FPGAs

This document describes the entire compilation path of CHAMPION. In Section 2, an overview of the flow is presented along with the process of incorporating a new function or module in CHAMPION.

The results obtained from implementing several applications using CHAMPION are then presented in Section 3. Sections 4 and 5 conclude and discuss possible extensions for this research work.

2. Technical Discussion

2.1 System Overview

The design flow of CHAMPION is shown in Figure 3. CANTATA is used as a function-oriented programming environment where all the application programs are developed using predefined functions called modules. Currently, a set of library modules has been developed in the CHAMPION project. A set of tools has been developed to automate the process of developing, verifying and installing the new modules in the CHAMPION library.

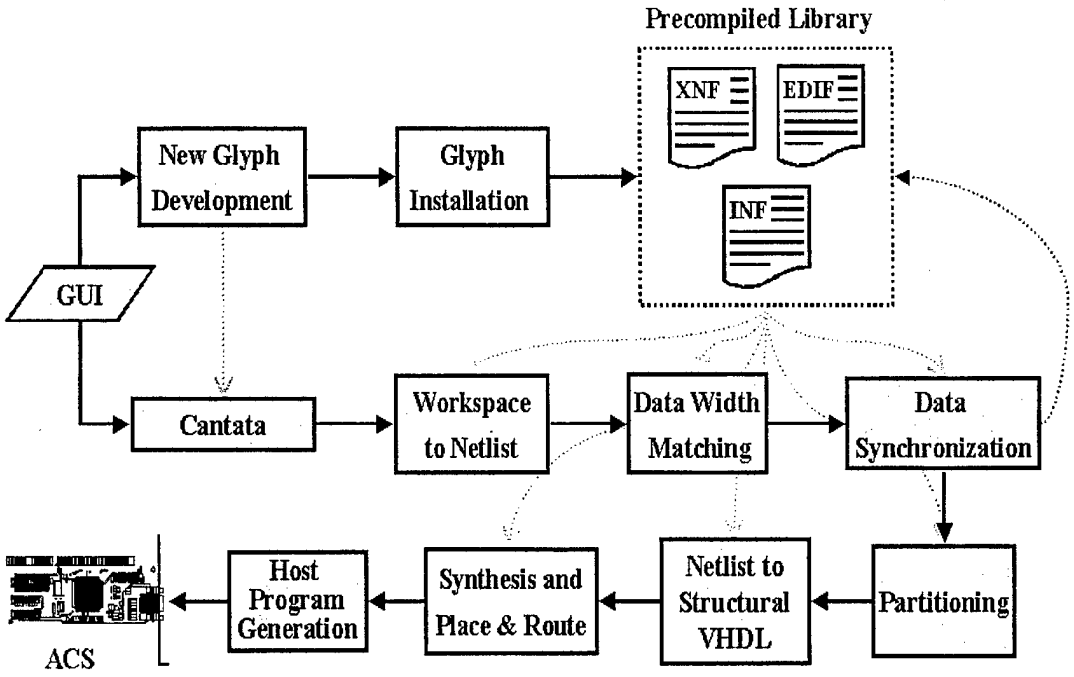


Figure 3. Overview of the CHAMPION Design Flow

Once the application is developed using CANTATA, the CANTATA program is translated into a more graph-oriented database, preserving the original modules and their interconnections. Then, each interconnection is checked to verify that the bit-widths of the connecting ports are the same.

Due to the difference in the processing time of each module, data traveling over different concurrent paths may arrive at the inputs of a multi-input module at different times. To ensure that each module generates the correct time-sequenced output, data synchronization is then performed. In CHAMPION, data synchronization is achieved by introducing delay buffers into the system. The synchronization software determines the lengths and locations of the delay buffers necessary to balance the various data

paths. An optimization algorithm is employed to calculate a set of buffer lengths and insertion points that maximizes the amount of buffer sharing, and therefore minimizes the total number of delay buffers.

Partitioning is then performed at the module-level, where each module element is represented by one node. This yields very low netlist complexity (hundreds versus tens of thousands). Therefore, the partitioning process has a very short runtime (seconds versus hours). Another advantage is that the functional flow information is preserved. Thus, debugging and simulation of the system are facilitated even after the partitioning.

After partitioning, the internal data structure or format is translated into a structural VHDL representation. The required I/O ports for each of the subnetlists are then added to the VHDL files. The VHDL files can then be synthesized and merged with the precompiled VHDL components corresponding to the CANTATA modules. Each subnetlist is then placed and routed. The resulting configuration files are downloaded to the corresponding programmable logic component on the ACS board.

In the next few sections, detailed descriptions of each component of the design flow are presented.

2.2 Library Cell Development and Verification

Application programs can be constructed by interconnecting CHAMPION modules using CANTATA. If certain modules needed for the application cannot be found in the precompiled CHAMPION library, these modules can be created and added. First, the designer must develop the fixed-point C or C++ program for the module. The reason for using fixed-point arithmetic is to allow the C/C++ program to mimic hardware operations. For complex functions, the C/C++ program can be formed as a macro of lower-level functions.

Next, VHDL code corresponding to each of the C/C++ programs must be generated. The functionality of the VHDL code must be identical to that of the C/C++ program. Identical test vectors are applied to both the C/C++ program and the VHDL code. The simulation results are compared to verify that bit-wise identical behavior is achieved. The steps for developing the new module are shown in Figure 4.

To accelerate the module development process, the commercial software, *A/RT Library* and *Builder* [9] from Frontier Design were integrated into the CHAMPION design flow. The *A/RT Library* and *Builder* provide the ability to generate the VHDL description of the hardware directly from a C-code specification of the application. The user no longer has to do this manually. The new steps for developing the module using *A/RT Library and Builder* are shown in Figure 5.

Once the functionalities are verified, the C/C++ program will be converted to a CANTATA module and installed in CANTATA using the tools from KRI. The corresponding VHDL description will be synthesized and converted to multiple technology-dependent netlist files (XNF and EDIF files) for multiple ACS boards. Also generated is a module information file (with extension *INF*) that stores different sizes, latencies and I/O data bit-widths of the module for multiple ACS architectures. The netlist and information files are then installed in the CHAMPION library as the hardware counterpart of the CANTATA module.

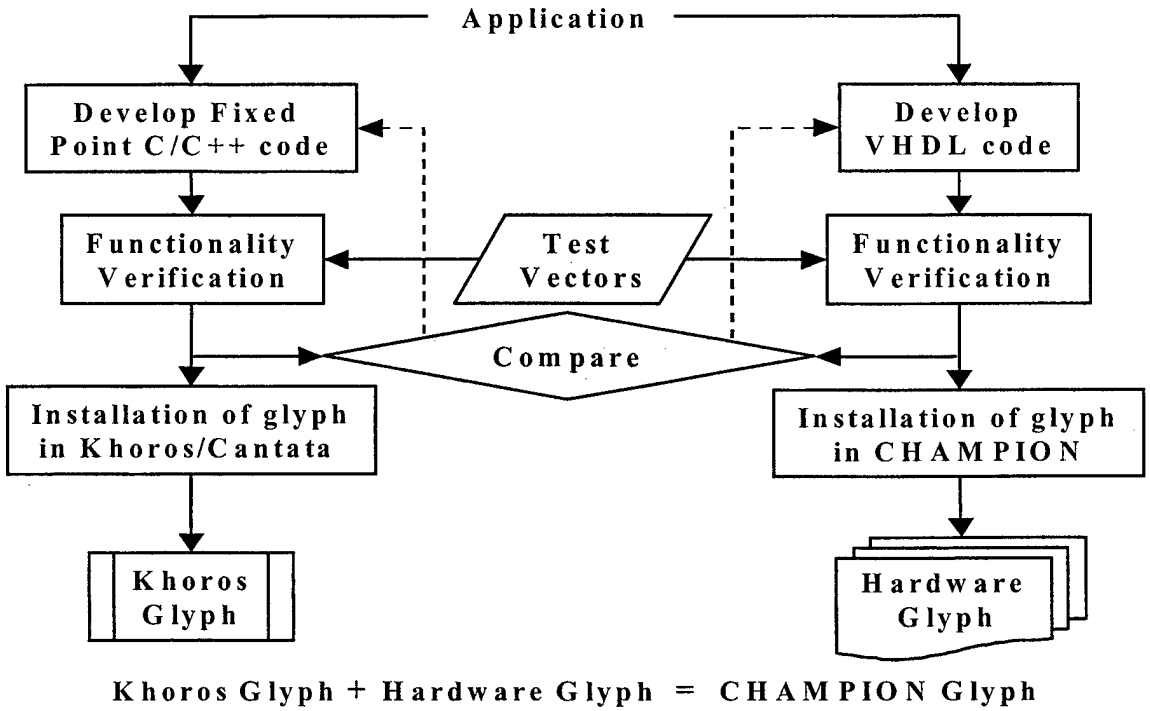


Figure 4. Steps for Developing a New Module

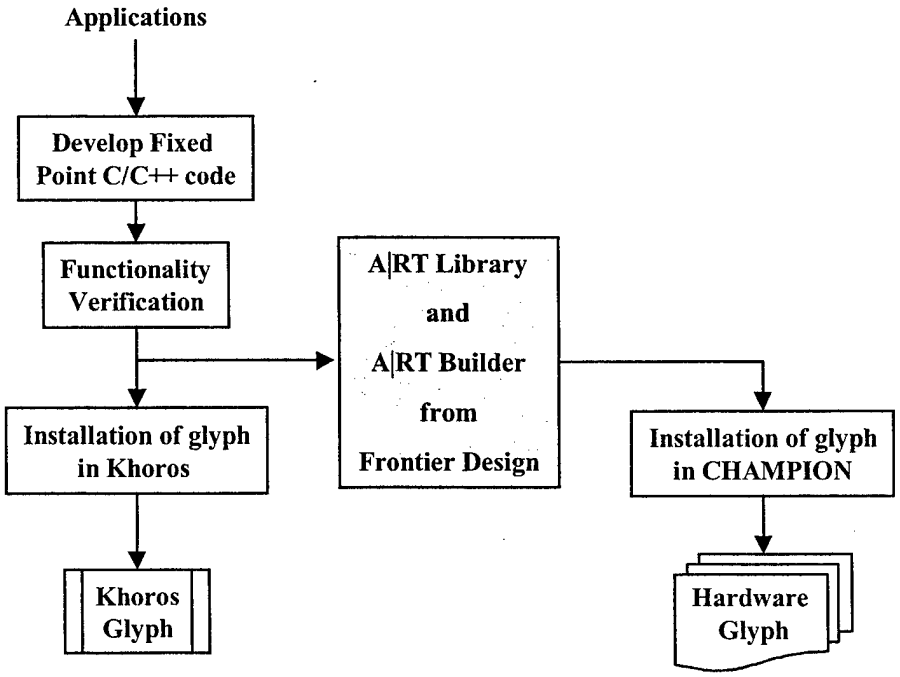


Figure 5. New Module Development Using A|RT tools

2.3 Converting CANTATA Workspace to Netlist

Using CANTATA, the designer can develop the application by interconnecting CHAMPION modules to form the CANTATA workspace. Simulation, data analysis and visualization can then be performed in CANTATA. Once the desired functionality of the application is achieved, the CANTATA workspace is translated into a directed hyper-graph where each module is represented as a node, and the interconnections between modules are represented as directed hyper-arcs. Based on the information from the *INF* file, weights are assigned to the nodes and hyper-arcs of the directed graph. This netlist format simplifies the use of graph theory and network optimization theory during the data synchronization and partitioning process.

2.4 Data Width Matching

In a hardware application, some functions may produce results that require fewer bits for their outputs than for their inputs. Consequently, cascaded modules may progressively require different data bit-widths. When one path of operations is connected to a parallel path, a mismatch in the number of bits for these inputs may occur. This mismatch is labeled *positive* since the bit-width of the net carrying the data is larger than the bit-width of the net receiving the data. An example of a positively mismatched data path is shown in Figure 6. A software tool within CHAMPION was developed to analyze each data path and to truncate the additional bits when appropriate. The truncating process (shown in Figure 7) is performed by inserting a "truncating" module at the mismatch data path. The "truncating" module will remove the additional data bits from the signal.

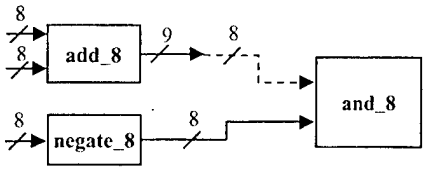


Figure 6. A Positively Mismatched Data Path

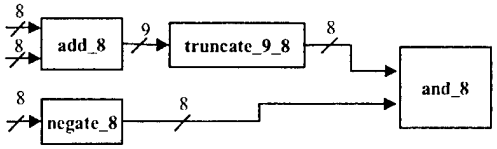


Figure 7. Insertion of a "Truncating" Module.

Similarly, some hardware functions may produce results that require more bits for their outputs than for their inputs, especially to avoid round-off errors. Consequently, a negatively mismatched data path such as the one shown in Figure 8 may occur. In this case, a "padding" module (shown in Figure 9) has to be inserted at the mismatched data path. The "padding" module will append 0's to the incoming signal.

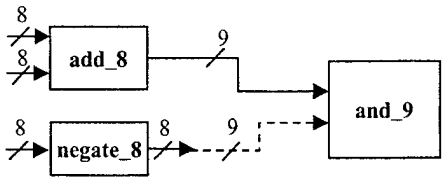


Figure 8. A Negatively Mismatched Data Path

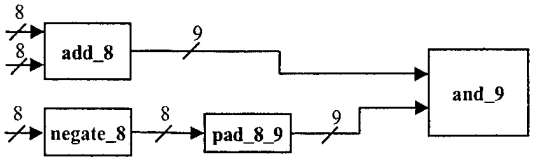


Figure 9. Insertion of a "Padding" Module

2.5 Data Synchronization

In a graphical programming environment such as CANTATA, executions of the programs are data driven. That is, each CANTATA module will begin execution only when all its input data is available. However, hardware systems are clock driven. At each clock cycle, each hardware module will process whatever data is presented at its inputs.

Due to the difference in the processing time of each hardware module, data traveling over different concurrent paths may arrive at the inputs of a multi-input module at different times. To ensure that each module generates the correct time-sequenced output, it is necessary that each module receive all its input data precisely at the same time.

Figure 10 illustrates this data synchronization problem. In this system, there are two primary input modules, two primary output modules and five processing modules labeled P_1 through P_5. The processing time for each processing module is labeled T_u. Note from the figure that the primary input and output modules have zero processing time. This is due to the fact that these modules are storage devices for the input and output data. No data processing will be performed in these modules. We assume that all the input data are readily available in the input modules. Therefore all input modules are time synchronized. We also require that all the outputs are made available at the same time. This is because if another digital system is connected to this system, these output modules will become the input modules for the new digital system. In addition, this requirement also allows us to break a large digital system into smaller subsystems for synchronization.

Examining this system, it is clear that the two input signals entering P_4 are not synchronized; the lower signal arrives one time unit sooner than the upper signal. Also, the primary output, O_2, becomes available two unit times earlier than O_1.

To synchronize this system, delay buffers can be introduced into the digital system. These delay buffers are inserted at various locations to delay the signal on the data path which has lower processing time, and therefore match it with the data path with higher processing time. For this synchronization approach, the pipelined time is zero. That is, the input signal can be presented to the system continuously without having to hold each new input signal, as in the case of the traditional synchronization approach using edge-triggered registers. This feature is essential in many designs which require high throughput rates.

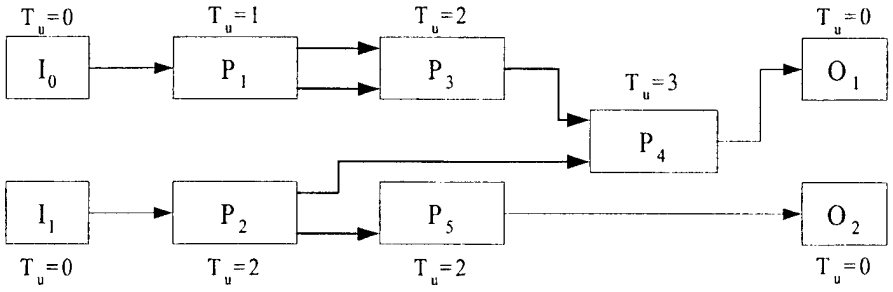


Figure10. An Unsynchronized Digital System

Synchronizing the system requires one to determine the lengths and locations of the delay buffers necessary to balance the various data paths. Straightforward methods for performing this task are not difficult to develop. For example, each multi-input processing module might be examined to check if all the inputs have the same delay from the primary input modules. If all the inputs do not have the same delay, the input with the largest delay is identified. For those inputs with delays which are less than this maximum value, delay buffers must be inserted into each of these "early" input lines in order for all the inputs to the module to have equal processing delay from the primary input modules.

Applying this method to Figure 10, we find that two delay buffers with a total of three unit time delays are required to synchronize the system (as shown in Figure 11). One buffer with a unit time delay has to be inserted in the path between P_2 and P_4. Another buffer with a two-unit time delay has to be placed between P_5 and O_1.

This straightforward method, however, does not necessarily provide the optimum solution in the sense that the total number of delay buffer units used is not necessarily a minimum. The delay buffers can always be moved forward or backward along the data path in order to achieve a maximal amount of delay buffer sharing. This can be seen in Figure 10 where inserting the delay buffer between I_1 and P_2 allows both the I_1-P_4 and the I_1-O_2 paths to share the delay. Therefore, a total of two unit time delays are required, compared to the three unit time delays in Figure 11.

An optimization algorithm can be used to calculate a set of buffer lengths and insertion points that maximizes the amount of buffer sharing and therefore minimizing total length of the delay buffers required. In the CHAMPION environment, the algorithm developed by Hu [10] was adopted and is described next.

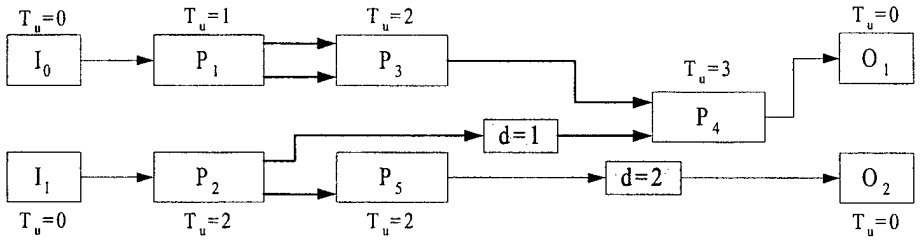


Figure 11. Synchronization Using the Straightforward Approach

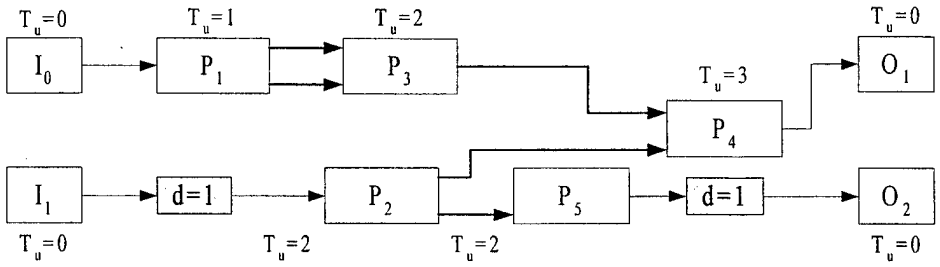


Figure 12. An Optimum Synchronization of Figure 11

2.6 Generating the Signal Flow Graph

To solve the buffer minimization problem using the algorithm proposed in [10], the system has to be represented using a signal flow graph (SFG) in which each processing module is represented as a node and each data path between the modules is represented as a directed edge. The weight of the edge directed from node u (corresponding to processing module P_u) to node v (corresponding to processing module P_v) is an unknown delay variable. This delay variable d_{uv} corresponds to the delay buffer which has to be inserted between P_u and P_v for synchronization. It will be determined during the synchronization process (solving of the delay buffer minimization problem). The value of d_{uv} computed during the synchronization process equals the size of the delay buffer required to be inserted between node u and node v.

Besides the nodes and edges that represent the modules of a system, two virtual nodes are introduced. One is termed the primary input node, while the other is called the primary output node. All of the input modules of the digital system will be combined and represented using the primary input node. Similarly, all the output modules of the digital system will be combined and represented using the primary output node. Therefore, in the SFG, all input signals to the system originate at the primary input node, and all outputs from the system terminate at the primary output node. Both of these special nodes are assumed to consume zero processing time.

Based on the rules stated above, a SFG representation can be constructed for any given digital system. For instance, for the digital system shown in Figure 10, the corresponding SFG is shown in Figure 13.

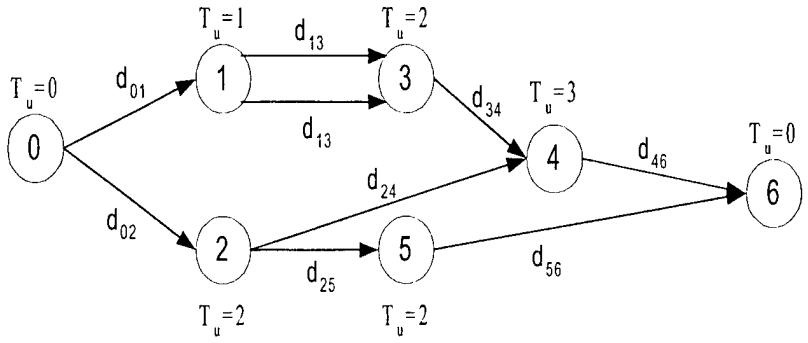


Figure 13. The SFG Representation of the Digital System Shown in Figure 10

The synchronizing problem is then reduced to assigning optimal values to the delay variables in the SFG.

An output connected to more than one input is called a hyper-arc (shown in Figure 14a). If hyper-arcs exist in the digital system, special representations of these nets need to be considered. One easy solution is to treat the hyper-arc as having multiple ordinary outputs and construct the SFG as shown in Figure 14b. The second representation can be obtained by inserting a virtual node, V1, with zero processing time as in Figure 14c. Another representation, which is proposed in [10], is shown in Figure 14d. This representation uses a binary tree structure which systematically introduces virtual nodes to the SFG.

The example in Figure 15 illustrates the effects of using the three different types of SFG representations in Figure 13. In the example, a total of 12 units of delay buffers are required to synchronize the hyper-arc if the SFG representation shown in Figure 14b is used. With the insertion of a virtual node, the total number of delay buffers can be reduced to six as shown in Figure 15b. If the binary structure is used, the total number of delay buffers can be reduced to five as shown in Figure 15c. This example demonstrates that some of the delay buffers can be shared along the hyper-arc through the insertion of the virtual node. The use of a binary tree structure in representing the hyper-arc can greatly exploit the buffer sharing property. To exploit the buffer sharing property fully, an algorithm must be used to arrange the processing nodes in the hyper-arc [10]. Different arrangements of the processing nodes driven by a hyper-arc may vary the total number of buffers along the hyper-arc because each arrangement may cause a different length delay buffer to be shared among the nodes driven by the hyper-arc.

2.7 Forming the Buffer Minimization Problem

To synchronize a system, all input signals to any given module must arrive at the same time. In other words, the accumulated delays along all the distinct paths from the primary input node, i, to a particular SFG node should be equal. The accumulated delay along a particular data path is simply the sum of all the weights of the edges and the processing time, T, of all the processing modules on the path from i to u in the SFG.

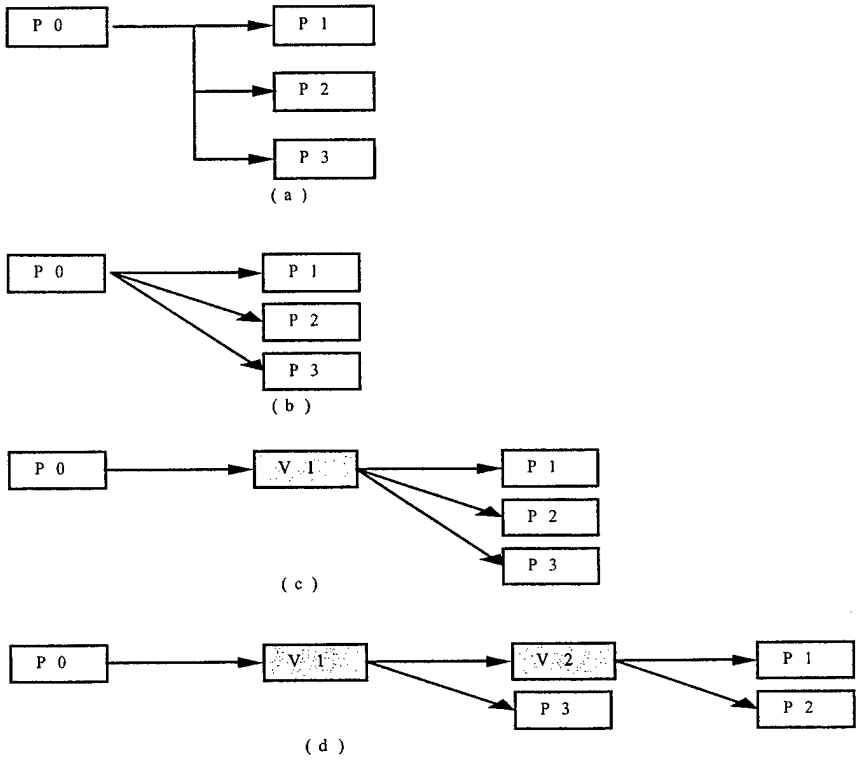


Figure 14. Different Representations of a Hyper-Arc

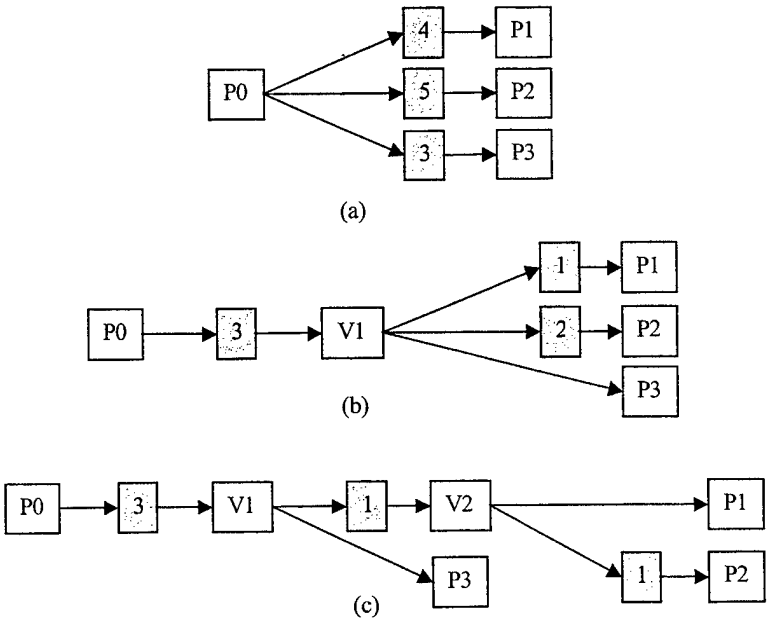


Figure 15. Delays of the Hyper-Arc

Denoting the j-th path between node i and u as $P_j(u)$ and the total delay along $P_j(u)$ as $D_j(u)$, it follows that $D_j(u)$ equals the sum of the delays associated with all of the edges along $P_j(u)$, i.e., it can be expressed as:

$$D_j(u) = \sum_{(x,y)\in P_j(u)} T_x + d_{xy} \tag{1}$$

where (x,y) represents an edge from node x to node y.

The synchronization problem can now be defined as assigning a value to each delay variable, d_{xy}, such that the values of $D_j(u)$ for $j = 1,2,\ldots k_u$ are identical for each and every u. Here, k_u denotes the number of distinct paths from the primary input to processing modules P_u.

Thus, the task of synchronizing a SFG may be seen to be equivalent to assigning values of D_u to all graph nodes and values d_{uv} to all graph edges such that

$$D_v - D_u = (T_u + d_{uv}) \tag{2}$$

holds for each and every pair of nodes, u and v, that are connected by an edge.

The synchronization-minimization problem can then be expressed as an ILP (integer linear programming) problem:

$$Mininmize \sum_{(u,v)} d_{uv} \tag{3}$$

$$Subject\ to: \ -D_u + D_v - d_{uv} = T_u \tag{4}$$

$$\begin{aligned} &u \in V, v \in V, (u,v) \in E \\ &D_u \text{ integer; } d_{uv} \geq 0, \text{ integer;} \end{aligned} \tag{5}$$

where E and V represent the sets of all edges and nodes in the SFG, respectively, (u,v) denotes a directed edge from node u to node v. The variables to be determined here are the D_u and d_{uv} values. The T_u's are known processing time delays of each node. Equations (3) through (5) form a standard description of an ILP:

$$\begin{aligned} &Minimize: d_{01} + d_{02} + d_{13} + d_{13} + d_{24} + d_{25} + d_{34} + d_{46} + d_{56} \\ &Subject\ to: \\ &\quad -D_0 + D_1 - d_{01} = 0 \\ &\quad -D_0 + D_2 - d_{02} = 0 \\ &\quad -D_1 + D_3 - d_{13} = 1 \\ &\quad -D_2 + D_4 - d_{24} = 2 \\ &\quad -D_2 + D_5 - d_{25} = 2 \\ &\quad -D_3 + D_4 - d_{34} = 2 \\ &\quad -D_4 + D_6 - d_{46} = 3 \\ &\quad -D_5 + D_6 - d_{56} = 2 \end{aligned} \tag{6}$$

2.8 Solving the Minimization Problem

In [10], it is shown that the constraint matrix of the ILP problem in equations (3) through (5) satisfies the definition of *unimodularity*. Therefore the ILP problem can be reduced to an LP problem [11]. The buffer synchronization-minimization problems can then be solved using a linear programming algorithm such as the Simplex method. Integer solutions are always guaranteed.

2.9 Partitioning

One of the main problems in partitioning is complexity. The research in partitioning theory has seen many algorithms with good results even with today's design complexity. The main disadvantage of these approaches is that they are based on gate-level netlists, thus requiring hours to execute.

In CHAMPION, we drastically reduced the complexity by keeping the structural information and configuring the programmable logic components and their interconnects in the ACS board into a linear array. With this topology, the partitioning operates on a module-level netlist and its order proceeds in a forward-only direction. Thus, partitioning is performed with netlists containing hundreds of nodes instead of tens of thousands.

For our design flow, the partitioning problem is based on the following constraints: capacity per partition, number of I/O pins per partition, RAM access, and temporal partitioning. The first two constraints are used to meet the limitations of the programmable logic components. The third constraint deals with the memory access for each programmable logic component. The architectures of some of the ACSs require that a fixed number of local RAMs are available to each FPGA for data writing and data reading. Therefore, a partition can contain only a certain number of RAM access modules. The fourth constraint deals with temporal partitioning of the ACS board. If the entire application cannot fit in one board configuration, then multiple configurations of the board are necessary and storage of intermediate results between board configurations is needed. In this case, one pair of RAM-access modules must be added to each configuration.

To solve the partitioning problem, three different approaches were investigated in our research. In the first and second approaches, we implemented two existing algorithms: a hierarchical partitioning (HP) method based on topological ordering [12] and a recursive partitioning (RP) algorithm based on the Fiduccia and Mattheyses bipartitioning heuristic [13]. Some modifications have been made on these algorithms to take advantage of the acyclic nature of our netlist, and to handle the RAM access constraint and the temporal partitioning constraint. A new recursive partitioning method based on topological ordering and levelization (RPL) [14] was also introduced. In addition to handling the partitioning constraints, the new approach efficiently addresses the problem of minimizing the number of FPGAs used and the amount of computation, thereby overcoming the weaknesses of the HP and RP algorithms.

After partitioning, the graph-based netlist for each partition is translated into structural VHDL. The VHDL files describing the ACS I/O ports and the precompiled VHDL components corresponding to the CANTATA modules are then merged with the structural VHDL. The resulting files are then synthesized, and then placed and routed separately.

The final step in the CHAMPION design flow is the generation of the host program which downloads each configuration file to the corresponding programmable logic component on the ACS. The host program also initializes the ACS board and reads the input data from the host workstation, sends the data to the ACS and writes output back to the host workstation.

The CHAMPION design flow allows CANTATA applications to be mapped onto multiple ACS hardware. In order to support different hardware architectures, multiple technology-dependent netlist files (XNF and EDIF files) were generated for each module in the CHAMPION library. Each module information file contains three sizes and latencies for the three ACS boards shown in Figure 16. Based on the ACS board selected by the designer, CHAMPION will select the corresponding technology-dependent netlist file and module information (size and latency).

The three ACS boards contain different programmable logic components. Therefore, for the partitioning problem, the capacity per partition, number of I/O pins per partition, RAM access, and temporal constraints are different for each board. Three constraint files were developed to store the constraints corresponding to the three ACSs.

The three ACS platforms also use different communication and control circuits, which are described using VHDL files specific for each ACS board. The ACS-specific file is integrated by CHAMPION into the structural VHDL files produced by the partitioning step. The resulting files are synthesized and then placed and routed.

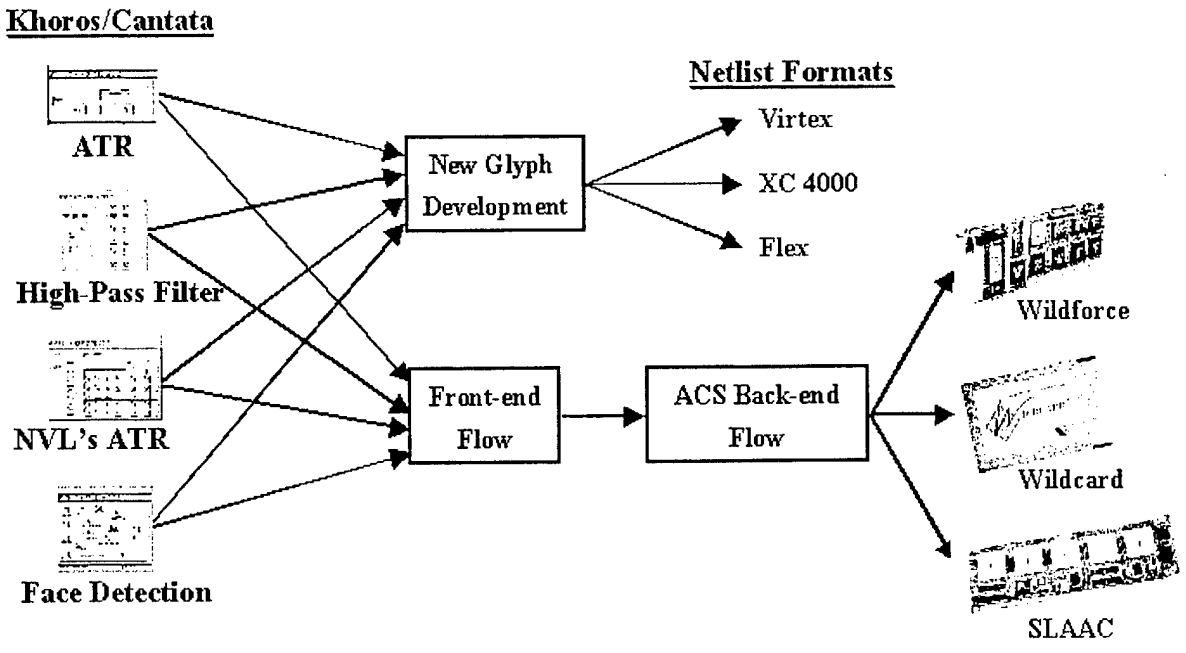


Figure 16. Mapping onto Multiple ACSs

It is particularly significant that CHAMPION can retarget to a new ACS board so quickly and easily. Given a high-level description of a new board that one may be considering designing or acquiring, a single-page data file is composed that lists the number of FPGAs, their size and I/O, the RAMs available and their interconnections on the board. The previously captured CANTATA application is then resubmitted to CHAMPION, which performs the mapping within an hour. Thus, application designers can determine which ACS architecture is the best match to the application and then either build or purchase the appropriate one.

This retargeting capability also permits an application designer to exploit the rapid advances in FPGA offerings. For example, as soon as a new ACS board is announced, the board-specific data file could be generated and the mapping/partitioning performed by CHAMPION. Upon arrival of the new board, the design could be downloaded and executed. The common situation of waiting months from the arrival of a new board until the application can be manually retargeted would be avoided.

3. Results

Verification of the CHAMPION design flow was conducted using four challenge problems: (1) a high-pass filter which detects edges in an input image, (2) an automatic target recognition algorithm (ATR) developed by Ben Levine of the University of Tennessee to identify multiple regions of interest in a set of images, (3) an automatic target recognition algorithm (Round-0) provided by the Army Night Vision Lab which involves template matching for tanks in images, and (4) a neural network based algorithm provided by NSA for detecting faces in images.

The high-pass filter application was captured in KHOROS/CANTATA using 18 modules interconnected by a total of 45 nets. It served as a simple test of the CHAMPION flow by moving an edge detection template over the entire input image to produce an output image with points corresponding to high-contrast edges, as shown in Figure 17. Each image required 14.2 seconds to process when executing in the CANTATA environment on a SUN UltraSparc. The same results were obtained using the Wildforce ACS platform in only 3.721 seconds. Most of the total hardware execution time consisted of transferring the image from the host CPU to the Wildforce board (2.669 seconds) with 1.049 seconds being required for configuring the FPGAs and only 0.003 seconds needed to process the algorithm itself. The entire CHAMPION design flow was executed in just 75 seconds, while structural synthesis and placement and routing required 2358 seconds.

The internally developed ATR algorithm [15-16] consisted of 93 modules interconnected by 226 nets. As illustrated in Figure 18, this algorithm takes as input infrared images and uses statistical methods to find probable targets such as tanks and armored personnel carriers, if present, and draws a box around any that are found.

Each image required 1054 seconds to process when executing in the CANTATA environment on a SUN UltraSparc running at 33 MHz. The same results were obtained using the Wildforce ACS platform in only 9.892 seconds. Most of the total hardware execution time consisted of configuring the FPGAs (9.147 seconds), with 0.735 seconds being required for transferring the image from the host CPU to the Wildforce board. Only 0.010 seconds were needed to process the algorithm itself.

Figure 17. Input and Output Images for the High-Pass Filter Application

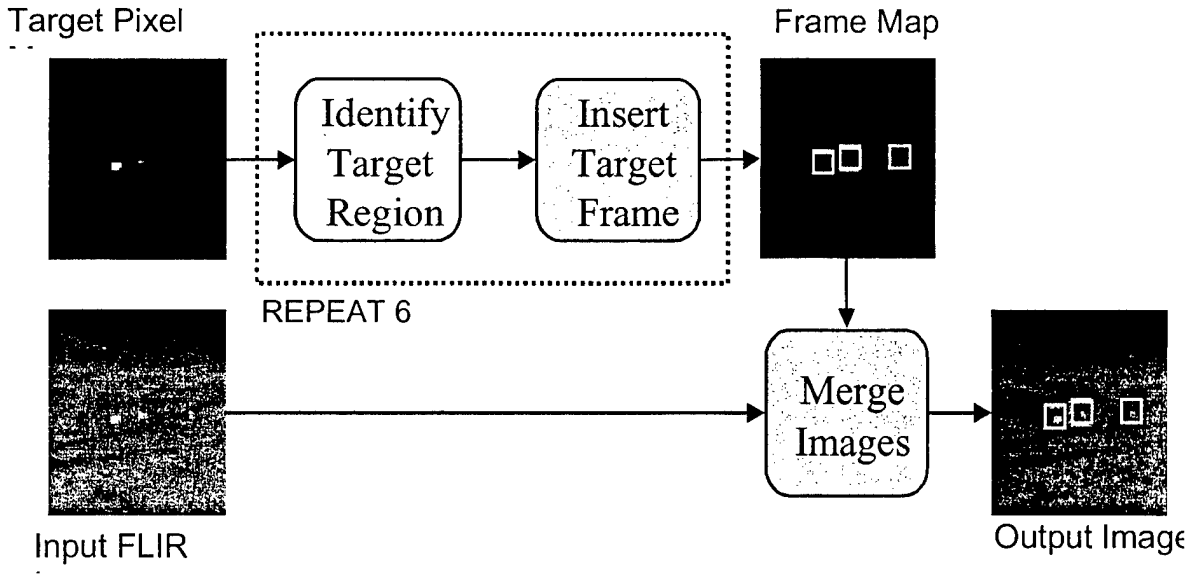


Figure 18. An Internally Developed ATR Algorithm

The entire CHAMPION design flow was executed in just 323 seconds, while structural synthesis and placement and routing required 13,378 seconds.

The ATR algorithm was implemented to serve as a benchmark for determining the improvement in mapping time for the design flow since it had been manually mapped to the Wildforce-XL board. The 250-hour manual process was performed automatically by CHAMPION in 5 minutes and 23 seconds, demonstrating a productivity improvement of over 2,000 times.

The ATR algorithm, Round-0, provided by the Army Night Vision Laboratory compares tank templates against the pixels of an input infrared image looking for matches. If identified, regions of interest are specified at the output. Additional rounds or iterations of the algorithm are required to complete the task of cueing an operator in the tank to these regions where his attention should be focused. However, Round-0 is the most computing intensive operation. Sample input and output images are shown in Figure 19.

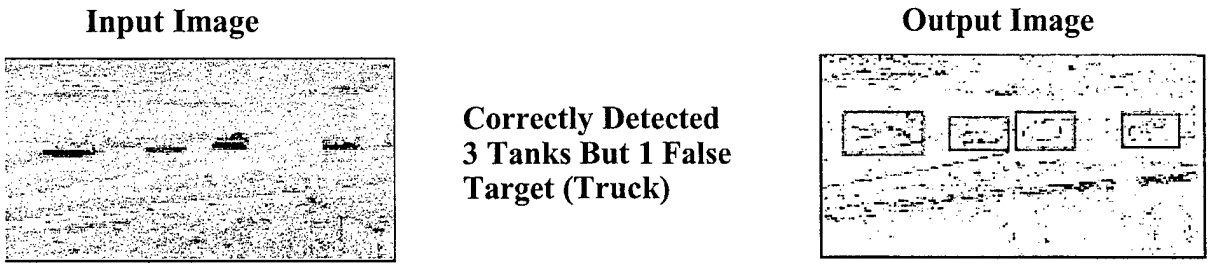


Figure 19. Input and Output Images for the Army Night Vision Lab ATR Algorithm

The CANTATA implementation of Round-0 consisted of 45 modules interconnected by 71 nets. Each image required 15,676 seconds to process when executing in the CANTATA environment on a SUN UltraSparc. The same results were obtained using the Wildforce ACS platform in only 154.076 seconds. Most of the total hardware execution time consisted of transferring the image from the host CPU to the Wildforce board (153 seconds) while it took only 0.807 seconds for configuring the FPGAs and only 0.269 seconds to process an image. The entire CHAMPION design flow was executed in just 24 seconds, while structural synthesis and placement and routing required 3,404 seconds.

NSA provided a neural network based algorithm for face detection that was originally developed at the Carnegie-Mellon University [17]. The input image is first processed on the host to locate the regions of interest in which faces are likely to be present. Then, the computationally intensive portion of the algorithm is executed on the ACS hardware platform. This section involves several neural networks which have a variety of predetermined weights. The output from the hardware is then processed by the host to merge and select the final result. Figure 20 shows an overview of the algorithm.

Three different neural networks were required: (1) UMEC, which has two hidden layers and one output layer to process a 30 by 30 window using a total of 5,882 connections; (2) Face17c, which has one hidden layer and one output layer to process a 20 by 20 window using a total of 2,905 connections; and (3) Face18c, which has one hidden layer and one output layer to process a 20 by 20 window using a total of 4,357 connections. At the time of this writing, the development of the modules required to implement this algorithm using CANTATA was not yet completed.

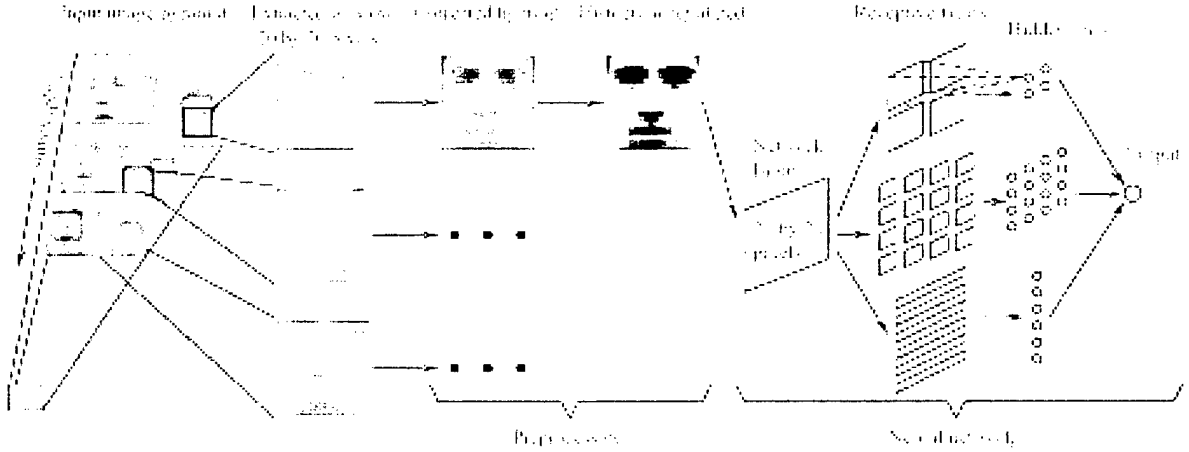


Figure 20. An Overview of the Face Detection Algorithm

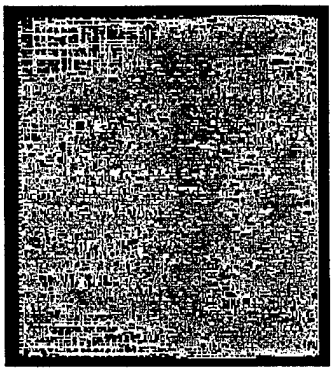
In addition to multiple ACS platforms containing multiple FPGAs, the CHAMPION flow was also used to target single-chip ASICs. The structural VHDL netlist produced by CHAMPION could be synthesized either on an incremental basis in which the hierarchy of the individual modules is preserved, or the hierarchical netlist could be flattened to produce a smaller layout at the expense of additional time for the synthesis step. Results were obtained for all of the applications described previously and are summarized in Table 1. Graphical views of the layouts are shown for one application in Figure 21. Low-cost prototyping of the Hewlett-Packard 3-metal, 0.5-micron CMOS process is available via MOSIS. Additional ASIC processes can easily be targeted with the appropriate commercial tools and technology files.

Table 1. Synthesis/PAR (Placement and Routing) Times and Chip Area for the ASICs

	Synthesis/PAR Time (secs)		Chip Area (mm^2)	
	Flattened	Hierarchical	Flattened	Hierarchical
High-Pass	24015	2958	13.7	14.9
Round-0	4647	8911	6.0	12.1
Umec	6728	6562	9.3	14.0
Face 17c	30818	6027	11.7	14.5
Face 18c	35448	6674	12.6	15.3
ATR	121697	51315	94.4	171.1

Total Synth/PAR Time = 14 min
Chip Area = 12.05 mm^2
Hierarchical Approach

Total Synth/PAR Time = 1 hr 17 min
Chip Area = 6.00 mm^2
Flattened Approach



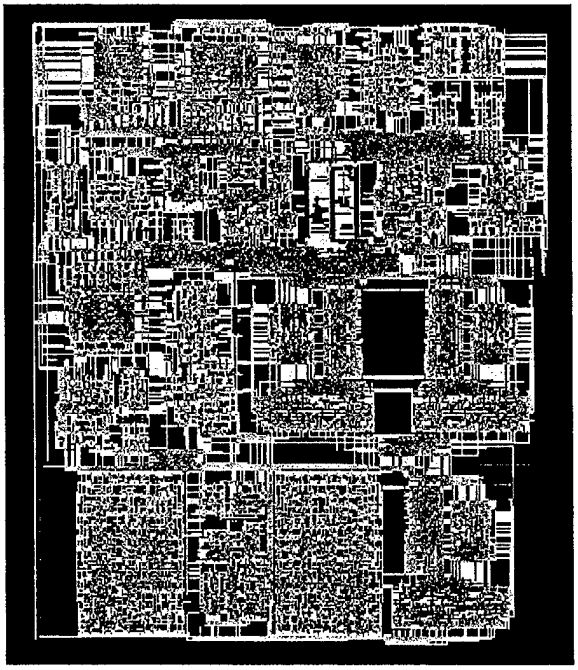


Figure 21. Graphical Views of the ASIC Layouts

4. Conclusions

In this document, a design flow for automatic mapping of CANTATA graphical applications onto multiple ACSs has been presented. Relative to contemporary technology, this design flow:

- Demonstrated a productivity improvement of 2,000 times over manual methods, as depicted in Figure 25.
- Utilized an optimization algorithm to synchronize the design using a minimum of well-placed delay buffers, and
- Partitioned the applications for multiple ACSs containing multiple FPGAs and provided a means to target multiple ACS hardware platforms as well as single-chip ASICs.

It is particularly significant that CHAMPION can retarget a new ACS board so quickly and easily. Given a high-level description of a new board that one may be considering designing or acquiring, a single-page data file is composed that lists the number of FPGAs, their size and I/O, the RAMs available and their interconnections on the board. The previously captured CANTATA application is then resubmitted to CHAMPION which performs the mapping within an hour. Thus, application

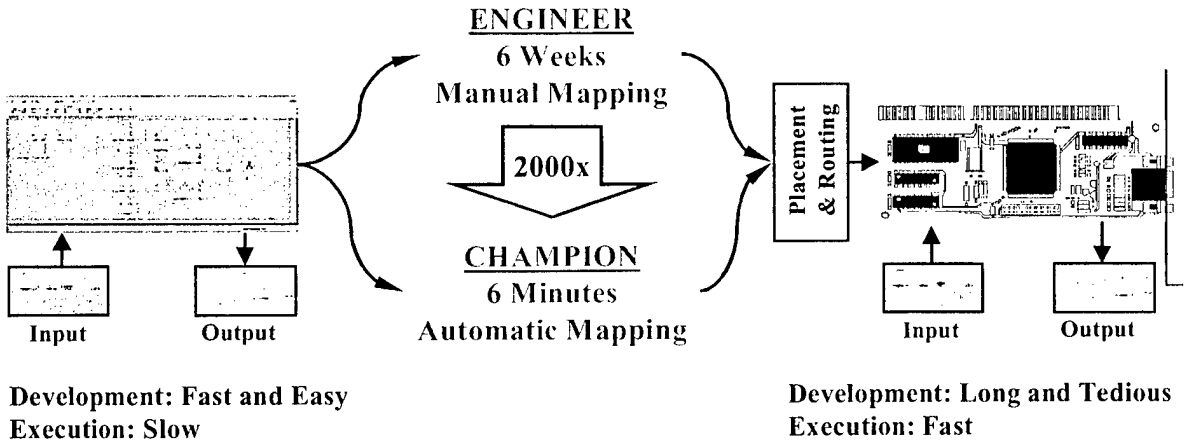


Figure 22. Design Time Productivity Improvement Provided by CHAMPION

designers can determine which ACS architecture is the best match to the application and then either build or purchase the appropriate one.

This retargeting capability also permits an application designer to exploit the rapid advances in FPGA offerings. For example, as soon as a new ACS board is announced, the board-specific data file could be generated and the mapping/partitioning performed by CHAMPION. Upon arrival of the new board, the design could be downloaded and executed. The common situation of waiting months from the arrival of a new board until the application can be manually retargeted would be avoided.

As indicated in Figure 23, CHAMPION has indeed been shown to make a significant impact on the development time for ACS platforms. However, extensive engineering effort is still required to master the specific hardware interfacing issues that must be fully understood for each ACS platform in order to get applications downloaded and executed on them. A standard format for expressing this information might be warranted to improve this situation. Then, each ACS vendor could provide a means of meeting this specification and thus reducing this portion of the development cycle.

Now that CHAMPION has been developed, an application described using CANTATA can be quickly mapped onto an ACS hardware platform. Designers can determine performance bottlenecks in data transfer or FPGA reconfiguration time and then either select a different ACS architecture or modify the CANTATA netlist. Numerous candidate solutions can be tried without the tedium of developing new VHDL descriptions. In essence, CHAMPION provides the means to reuse and integrate VHDL modules. The graphical entry and simulation capabilities provided by CANTATA facilitate this process.

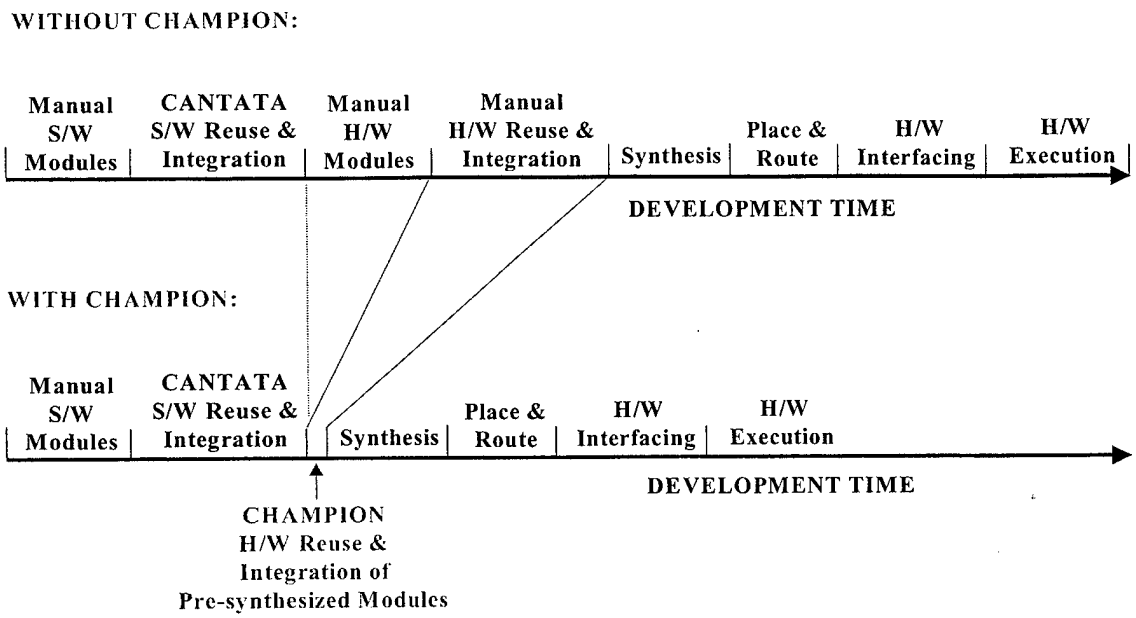


Figure 23. Impact of CHAMPION on Development Time

Four publications [15, 18-20] resulted from this project and were presented at the major conferences in this field. Also, five theses were written. Ben Levine obtained his M.S. and is now pursuing a Ph.D. at Carnegie Mellon University in Pittsburgh. Senthil Natarajan completed his M.S. and is now employed at Motorola Research Labs outside Chicago. Nabil Kerkiz obtained his Ph.D. and is working with Intel in Santa Clara, while Sze-Wei Ong finished his Ph.D. and has interviewed with Intel in Portland. Bernadeta Srijanto should complete her M.S. in a few months.

5. Recommendations

In terms of extending CHAMPION, one possibility is to permit it to accept inputs from other graphical programming environments such as LabVIEW from National Instruments and/or Simulink from MathWorks. To include these programming environments, the only component in the design flow that needs to be modified is the front-end translator that converts the CANTATA workspace into a CHAMPION netlist. The rest of the steps in the design flow will remain the same.

6. REFERENCES

[1] J. Rasure and S. Kubica, *The KHOROS Application Development Environment*, KHOROS Research Inc., Albuquerque, NM, http://www.khoral.com.

[2] D. Argiro, S. Kubica, and M. Young, "CANTATA: The Visual Programming Environment for the KHOROS System" Khoral Research, Inc., Albuquerque, NM, http://www.khoral.com.

[3] Annapolis Micro Systems, Annapolis, MD, http://www.annapmicro.com.

[4] J. Babb, M. Frank, V. Lee, E. Waingold, R. Barua, M. Taylor, J. Kim, S. Devabhaktuni, and A. Agarwal, "The RAW Benchmark Suite: Computation Structures for General Purpose Computing," *Proc. IEEE Symposium on FPGAs for Custom Computing Machines,* Napa Valley, CA, April, 1997.

[5] N. Ratha, A. Jain, and D. Rover, "Convolution on Splash 2," *Proc. IEEE Symposium on FPGAs for Custom Computing Machines,* Napa Valley, CA, April, 1995.

[6] Systems Level Applications of Adaptive Computing (SLAAC), http://www.east.isi.edu/projects/SLAAC.

[7] P. Banerjee, et al., "A MATLAB Compiler for Distributed Heterogeneous Reconfigurable Computing Systems," *Proc. IEEE Symposium on FPGAs for Custom Computing Machines,* Napa Valley, CA, April, 2000.

[8] J. Hammes, R. Rinker, W. Böhm, W. Najjar, B. Draper, and R. Beveridge, "Cameron: High Level Language Compilation for Reconfigurable Systems," *Conference on Parallel Architectures and Compilation Techniques*, Newport Beach, CA, Oct. 12-16, 1999.

[9] D. Johnson, "Architectural Synthesis from Behavioral C Code to Implementation in a Xilinx FPGA," *Business Development Manager, Frontier Design Inc.,* http://www.frontierd.com/

[10] X. Hu, S. C. Bass and R. G. Harber, "Minimizing the number of delay buffers in the synchronization of pipelined systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 12, pp.1441-1449, December 1994.

[11] A. J. Hoffman and J. B. Kruskal, "Integral boundary points of convex polyhedra," *Linear Inequalities and Related Systems,* ed. H. W. Kuhn and A. W. Tucker, Princeton, N.J.: Princeton University Press, pp. 223-46, 1956.

[12] B. Stanley, "Hierarchical Multiway Partitioning Strategy with Hardware Emulator Architecture Intelligence," Georgia Institute of Technology, Ph.D. Dissertation, 1997.

[13] R. Kuznar and F. Brglez, "PROP: A Recursive Paradigm for Area-Efficient and Performance Oriented Partitioning of Large FPGA Netlists," *International Conference on Computer-Aided Design*, pp. 644-649, November 1995.

[14] Kerkiz N., "Development and Experimental Evaluation of Partitioning Algorithms for Adaptive Computing Systems," University of Tennessee, Ph.D. Dissertation, 2000.

[15] Levine, B., Natarajan, S., Tan, C., Newport, D. and D. Bouldin, "Mapping of an Automated Target Recognition Application from a Graphical Software Environment to FPGA-based Reconfigurable Hardware," *Proc. IEEE Symposium on FPGAs for Custom Computing Machines,* Napa Valley, CA, April, 1999.

[16] Levine B., "A System for the Implementation of Image Processing Algorithms on Configurable Computing Hardware," University of Tennessee, Masters Thesis, 1999.

[17] H. Rowley, S. Baluja and T. Kanade, "*Neural Network-Based Face Detection*," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, No. 1, pp. 23-38, January 1998.

[18] S. Ong, N. Kerkiz, B. Srijanto, C. Tan, M. Langston, D. Newport, and D. Bouldin, "Automatic Mapping of Multiple Applications to Multiple Adaptive Computing Systems," *Proceedings of 2001 IEEE Symposium on Field-programmable Custom Computing Machines (FCCM)*, Rohnert, CA, April 30, 2001.

[19] S. Ong, N. Kerkiz, B. Srijanto, C. Tan, M. Langston, D. Newport, and D. Bouldin, "Design Flow for Automatic Mapping of Graphical Programming Applications to Adaptive Computing Systems," *Proceedings of the High Performance Embedded Computing Workshop (HPEC)*, Boston, MA, Sep. 23, 2000.

[20] S. Natarajan, B. Levine, C. Tan, D. Newport and D. Bouldin, "Automatic Mapping of KHOROS-based Applications to Adaptive Computing Systems," *Proceedings of 1999 Military and Aerospace Applications of Programmable Devices and Technologies International Conference (MAPLD)*, pp. 101-107, Laurel, MD, Sept. 28-30, 1999.

7. LIST OF ACRONYMS

ACRONYM	DESCRIPTION
ACS	Adaptive Computing Systems
AFRL	Air Force Research Laboratory
AMS	Annapolis Microsystems Inc.
ASICs	Application-Specific Integrated Circuits
ATR	Automatic Target Recognition
CMOS	Complementary Metal-Oxide Semiconductor
DARPA	Defense Advanced Research Projects Agency
EDIF	Electronic Data Interchange Format
FPGA	Field-Programmable Gate Array
HP	Hierarchical Partitioning
ILP	Integer Linear Programming
LP	Linear Programming
MOSIS	Metal-Oxide Semiconductor Implementation System
NVL	Night Vision Laboratory
PAR	Placement And Routing
RP	Recursive Partitioning
RPL	Recursive Partitioning with Levelization
SFG	Signal Flow Graph
SLAAC	Systems Level Applications of Adaptive Computing
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Scale Integrated Circuit
XNF	Xilinx Netlist Format