Second Semiannual Technical Report

TRANSFORMATION of ADA PROGRAMS INTO SILICON

82 Mar 1 – 82 Oct 31

Elliott I. Organick, Principal Investigator
(801) 581–6087

Contractor: The University of Utah

Date of Contract: 81 SEPT 1

Expiring: 83 AUG 31

Sponsored by

Defense Advanced Research Projects Agency (DoD)
ARPA Order No. 4305

Under Contract No. MDA 903–81–C–0411, issued by
Defense Supply Service – Washington, Washington DC 20310

November 1982

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: UTEC-82-020	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: TRANSFORMATION OF ADA PROGRAMS INTO SILICON	5. TYPE OF REPORT & PERIOD COVERED: semi-annual, 81 Sept 1 - 82 Feb 28
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Dr. E. Organick, Dr. G. Lindstrom, D. K. Smith, Dr. Subrahmanyam, T. Carter	8. CONTRACT OR GRANT NUMBER(s): MDA 903-81-C-0411
9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of Utah, Computer Science Department, Salt Lake City Utah 84112	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 1001/1122
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency (DoD), 1400 Wilson Boulevard, Washington, D.C. 22209	12. REPORT DATE: March 1982
	13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: Defense Supply--Service Washington, Rm 1d-245, The Pentagon, Washington, D.C. 20310	15. SECURITY CLASS. *(of this report)*: unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

control unit, CADDET, SPICE Internet protocol, submodules, Ada-to-silicon, transformation methodologies, high level program specifications, DoD Internet Protocol, special function architecture, ADA packages & tasks, VLSI synthesis, program formal specifications, device modeling, switched capacitor filter, stored logic array, logic simulator, hand shake, speed-independent, one-hot, portable standard LISP, silicon compiler, VLSI

This report outlines the beginning steps taken in an integrated research effort toward the development of a methodology, and supporting systems, for transforming Ada programs, or program units, (directly) into corresponding VLSI systems. The time seems right to expect good results. The need is evident; special purpose systems should be realistic alternatives where simplicity, speed, reliability, and security are dominant factors. Success in this research can lead to attractive options for embedded system applications.

DD FORM 1 JAN 73 1473

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

Ada programs can be regarded as ensembles of machines, one per program unit (module), which in turn may be mapped directly into corresponding VLSI structures on one or more chips with interconnecting (packet switched or other) communication nets.

Many of the transformation steps, when performed manually, when optimization is not everywhere crucial, and when care is taken to constrain somewhat the structure of the source Ada program, appear to be understood.

The research reported here is part of a five-year plan, the first year of which focuses on "proving" the concepts through a realistic demonstration of methodology for a specific example Ada program (a silicon representation of part or all of the DoD Standard Internet Protocol, IP, initially expressed in Ada.) Since the mapping from Ada to VLSI is seen as a multistep, iterative procedure, considerable effort for the following four and a half years will be the invested in the development and tailoring of intermediate languages and their bridging algorithms (compilers), as needed, and in the development of objective criteria for their use with feedback loops for iterative design.

Implicit in these objectives is the development of a set of hardware structuring paradigms (rewrite rules) whose application can ensure that transformation steps between levels of abstraction in the design process are well structured in order to preserve the integrity and, where possible, the clarity of the original Ada specification. Some paradigms, but of course not all, lead to highly efficient implementations.

Abstract

This report summarizes the second six months of work of the coordinated research project, "Transformation of Ada Programs into Silicon." (The main objectives of this project were outlined and then introduced in depth in the preceding semiannual report.) In the past seven months, work has advanced in three main areas. Expanded summaries of work in these areas (and subareas) are presented:

1. Work on the principal case study of this project: Converting the DoD Internet Protocol to silicon. The full Protocol has been decomposed into three main parts. The part that handles outbound datagrams has been fully specified in Ada and an interesting part of that code has been transformed into an NMOS circuit composite represented in PPL (Path Programmable Logic).

2. A tranformation system is being implemented to map Ada program units into intermediate forms in syntactically correct Ada. These intermediate forms are suitable for input to the transformation system (ASSASSIN) that automates the production of the asynchronous control components of the PPL circuit composites. A theory for synthesizing circuits from system specifications that are more abstract than Ada is also reported.

3. Research and Development on the design, fabrication, and application of PPL (Path Programmable Logic) circuit arrays is reported

    a. The ASSASSIN system which transforms state graphs of state machines expressed in textual form to self-timed PPL programs and composites is operational.

    b. Completion of a PPL simulator (ASYLIM) has been incorporated into the PPL design system.

    c. Design and composite layout of three different PPL test circuits were sent out for fabricaton. The circuits will be used to check a wide variety of PPL cells and supporting circuitry.

    d. A design technique for ICs representing self-timed stored state machines and data path components using the PPL cell set has been developed. The results of the research have produced new PPL macro cells which augment the set of available cells.

1. Summary

This report summarizes the second six months of work of the coordinated research project, "Transformation of Ada Programs into Silicon." Project objectives span a broad and ambitious spectrum (broader than the already broad title implies), hence the term *coordinated*; this refers to the fact that, on the one hand, all research within the project is closely related, but that the overall project success is not predicated on close coupling of individual subproject results. The main objectives of this project were outlined and then introduced in depth in the preceding semi-annual report [19]. They are repeated here in more brief and in a somewhat updated form:

1. Develop elements of a tranformation methodology for converting Ada programs or their parts, into VLSI systems. This research includes identifying a sufficient set of transformation rules for mapping program specifications through successive levels of representation, from Ada or related abstract specifications, to integrated circuits.
2. Demonstrate the methodology developed in 1 by manually applying it to a non-trivial example: transforming an Ada-encoded representation of the DoD Standard Internet Protocol [20] (or a significant subset thereof) into NMOS circuitry.
3. Work toward a theory for identifying substructures within Ada programs for which the transformation methodology is pragmatically attractive.
4. Develop specifications for a set of software tools for use in automating the transformation methodology developed in 1.
5. Develop a methodology for testing integrate circuits representing Ada program units and for integrating such circuits into a larger system.

In the past seven months, our work has advanced in three main areas and in several subareas listed below. Expanded summaries of work in these areas are presented in succeeding sections of this report.

1. Work on the principal case study of this project: Converting the DoD Internet Protocol to silicon. The full Protocol has been decomposed into three main parts [18, 13]. The part that handles outbound datagrams has been fully specified in Ada [14] and part of that code has been transformed into an NMOS circuit composite [6].
2. Implementing a tranformation system to map Ada program units into intermediate forms in syntactically correct Ada. These intermediate forms represent <state machine, data path> pairs suitable for input to another transformation system that automates the production of circuit composites [24].
    a. Development of a theory for synthesizing circuits from system specifications that are more abstract than Ada, e.g., axiomatic algebraic specifications or from Ada augmented with ANNA-like specifications that also allow specification of temporal properties. [12, 29, 25, 26]
3. Research and Development on the design, fabrication, and application of PPL (Path Programmable Logic) circuit arrays.
    a. Completion of the transformation system called ASSASSIN, reported in detail elsewhere [7], which transforms state graphs of state machines expressed in textual form to self-timed PPL Programs and composites.
    b. Design and composite layout of three different PPL test circuits called UU20, UU21, and UU23. UU20 is used to check the read-enable flip-flop, the write-enable flip-flop, the asynchronous-clear flip-flop, row pass-transistors, and flip-flop pull-up cells. UU21 checks the Set/Reset flip-flop, the two-wire latch, the inverter cells, the column pass-transistor, and the S, R, 1, and 0 cells. UU23 checks the input and output pad cells. In addition, a test circuit containing several different oscillators and counters has been included for determining performance.

       UU20 and UU21 were sent to MOSIS for the June 4 run, and in July we were informed that, due to some mask problems, none of the circuits were completed. We are still waiting for these parts. In September we decided

to process all three test circuits in our own (HEDCO) laboratory. Problems with mask making equipment have caused delays, however, UU20 and UU21 are expected out of the process line in late November or early December. UU23 should also be processed in December.

c. Completion of a PPL simulator called ASYLIM which has been under development for the past year. (The work was sponsored primarily by a commercial company. The simulator was incorporated into the PPL design system for use in this project. The main characteristics of this simulator are outlined in Section 4 of this report.

d. Development a design technique for ICs representing self-timed stored state machines and data path components using the PPL cell set. (The work was sponsored by a private company.) These techniques have been primarily directed at the design of circuits using a conventional single-rail Four Cycle signalling protocol. The results of the research have produced new PPL macro cells which augment the set of available cells.

2. Converting the DoD Internet Protocol to Silicon.

by

Elliott I. Organick and Gary Lindstrom

As mentioned previously [19], our design of the Protocol is based on a decomposition into three submodules: INM_OUT dealing with traffic outbound on a given local net, INM_IN similarly handling inbound traffic, and INM_SRV tying them together and interfacing to the Host(s). We envision one INM_IN and INM_OUT pair of submodules for each local net interface, but only one INM_SRV submodule per Internet Module (INM).

We are following the five-level software development and testing plan discussed in the preceding report. The levels correspond to IP applications in increasingly generalized settings. The plan stipulates testing as each level is reached, rather than as an epilog to the development plan. Testing is to be conducted at several levels, from the physical characteristics of the circuits themselves to the (Ada) semantic behavior of the submodules that have been converted to circuits.

After designing (specifying) the interfaces between the submodules [13, 10], we then selected the INM_OUT (sub)module as the first one to be converted to circuitry. Work toward this objective in the past seven months has been rapid in some respects and slow in others.

The specific and significant accomplishments have been as follows:

1. We have coded the complete INM_OUT submodule in Ada and have succeeded in compiling most of it for execution on the Intel iAPX 432 system except for statements and declarations associated with uses of the Ada rendezvous construct.

   [As later versions of the Intel compiler become available, we expect not only to be able to compile the full module using rendezvous syntax and semantics, but to execute it in this mode as well. In the meantime we are working with a version of the code that simulates each rendezvous via Send/Receive primitives instantiated through use of the Ada generic package mechanism.]

2. The INM_OUT submodule is an Ada package named INM_OUT_Module; it contains three intercommunicating Ada tasks. We are in the process of transforming each of these tasks into PPL circuit composites beginning with the second one listed below:

   a. The main task, named INM_OUT, interfaces with INM_SRV and with LNM_OUT such that a pipeline effect is achieved for speeding datagrams along the outbound data path: Host module —> INM_SRV —> INM_OUT —> LNM_OUT.

   b. An auxiliary (server) task, named Read_Init_Parameters, which obtains from host-related memory the initial parameter values needed to perform datagram transmission. Transformation of this server task, one which is rich in Ada control structures, is essentially completed. A demonstration, showing the process by which we make the transformation to PPL circuit composite was given in June, 1982 during a DARPA review of our project. That demonstration was based on a preliminary version of the Ada task, which has now been updated. The composite produced for the current version of the task is more interesting and is apt to resemble more closely the one we eventually will consider the final version.

   c. An auxiliary task named Translate_TOS_Task, which operates in parallel with INM_OUT, the main task, by translating type-of-service information from host-level to local-net level encoding.

3. As just mentioned, the task Read_Init_Parameters has now been converted semi-automatically to PPL circuit composites in NMOS. The conversion into PPL composite form is discussed in part in a new paper by Carter, to be presented at a DARPA-sponsored meeting at Stanford, on November 5 and in part below. Carter's paper focuses primarily on the technology for converting the control structure portion of the Ada task into the self-timed control unit of the

corresponding circuit.

In this report we make some observations on the overall structure of Read_ Init_ Parameters and on some of its subtle details. We also comment on some of the steps we traversed in arriving at this version of the task. A copy of the body part for the present version of this Ada task is to be found in the Appendix.

[The complete Ada specification of the INM_OUT submodule, which includes this task is given in a separate report [14]. A reader of the Appendix version only is expected to imagine how the task Read_ Init_ Parameters interfaces with the remainder of the entire submodule. A reader of the separate report is treated to a "road map" of the full Ada structure of the INM_OUT submodule which helps to understand our overall design.]

4. As a prelude to testing hardware versons of Ada pargram units and in support of our work in specifying subsystems in Ada and then simulating them, we installed, made operational, and have begun using a complete Intel 432 Cross Development System. This system includes an Ada cross compiler for a large subset of Ada and a 432 multiprocessor system consisting of two regular and two interface processors. We expect to receive from Intel a compiler that includes full tasking by the end of calendar 1982 and an equally complete resident compiler approximately a year later. We have also gained hands-on familiarity with a number of the 432 System's operating system features.

2.1. Interesting aspects of Read_ Init_ Parameters

The structure of Read_ Init_ Parameters includes a number of typical and interesting features of Ada tasks both from the point of view of inter-task communication and intra-task body structure.

- Inter-task communication. The task includes nested accept statements both of which have both in-bound and out-bound parameters. There accept statements are implemented using simple request/acknowledge protocols.
- Intra-task computation. The task body includes a rich nested loop structure and one nested block defining local variables whose ranges are determined dynamically. The loops include the infinite outermost loop of the task, familiar "for" loops with fixed upper bounds, and indefinite loops escapes from which are based on "exit when" clauses. As we have expected all along, all of these Ada control structure forms map in a straightforward way to corresponding control structures at the state machine level and thence to PPL circuits.

The data path of Read_ Init_ Parameters includes several variables which are represented in the hardware as registers or counters. One array variable is represented as a RAM to represent a map from type-of-service encoded at the host level to type-of-service encoded at the local net level. [The size of this RAM, which is never apt to be very large in any case, is limited to four-octets (for a 2 by 2 array) in our demonstration implementation. Most of the above variables are shared with the other two tasks of the submodule; that is, they are declared local to the containing package, INM_OUT_Module, however we perceive no difficulty in achieving mutually exclusive access.

The one variable that is local to the entire server task does not and is not represented in hardware as a storage element. Variables used locally for loop control are represented as hardware counters and/or registers, but some sharing is achieved where there is no chance for conflict.

Although the transformation to the Ada code to the "engine level", i.e., to representation as a (control unit, data path) pair, has been done by hand, the transformation research reported in the next section has included consideration of each of the "hand-made" mapping steps in this particular exercise.

2.2. Arithmetic processing

That we have encountered so little trouble performing the mapping for this task is partially explained by the fact that the task involves only trivial arithmetic processing. (Indeed, the entire INM_OUT_Module involves only minor arithmetic processing.) At this stage of our research we are glad this is the case as we consider it important to determine first what new challenges, if any, must be met for achieving asynchronous control.

2.3. On going and future related work

Now that this part of the research is essentially complete, including the development of the ideas embodied in ASSASSIN, we expect to be concentrating next on such challenges as the application of the same or related asynchronous design principles to arithmetic processing. Also included in our agenda is research intended to help us automate the mapping of data path storage components, identified in the transformation from Ada program units, into PPL circuits coupled to their controls.

3. A Transformation System: Theory and Implementation

by

P.A. Subrahamanyam

We have made substantial progress along two directions: implementation of a prototype transformation system and further development of a conceptual/theoretical basis to support the design of integrated software–hardware systems. We outline the major contributions below, with appropriate pointers to references that contain more detailed discussions.

3.1. Systems Implementation

- A set of tools to support experimentation with Ada–to–Silicon transformations has been implemented, and runs on the TOPS–20. The system has been ported to the VAX–750, and an initial version has been installed. This porting proved to be a major job (and problem) due to unstated incompatibilities between INTERLISP–20 and INTERLISP–VAX. Further debugging and testing of the Vax version will be done when the experimentation is moved completely over to the Vax. (Given the needed personnel, we expect this to be carried out over the next year, when our address space requirements force us to move over to the Vax).
- An initial set of transformation routines has been implemented and is being augmented so as to handle additional syntactic constructs in Ada. This set of programs is intended to aid in the interactive generation of the target hardware description in a symbolic representation. Details of the current status of this work are reported in [24].

3.2. Conceptual/Theoretical Basis for Transformation

- A unified theoretical framework to support a broad spectrum of the VLSI design process has been introduced in [29], which is currently available in the form of the draft of a research monograph. This monograph introduces an algebraic framework to aid in the synthesis and verification of special purpose VLSI systems, proceeding from high level specifications. It allows for abstract specifications of the syntax, semantics, temporal and performance requirements particular to a given problem. The characteristics of the environment in which the system is embedded can also be specified and are used in the synthesis process. In addition, the framework allows several of the constructs in existing languages to be modelled, including nondeterminism, concurrency, and data/demand driven evaluation. This allows the infrastructure to be (1) applied to situations wherein the problem "specification" is in the form of a program in a conventional high level language and (2) used to model the lower level synchronous/asynchronous nature of implementations. Topology and circuit layout geometry can also be expressed by using the algebraic primitives available.
- Annotations to Ada have been proposed to aid the abstract specification of temporal properties of systems and desired performance requirements [25, 28, 12].
- Transformation methods to apply the theory in the context of Ada to obtain systolic implementations are detailed [27, 24].
- An algebraic modelling of weak conditions to be met by asynchronous circuits has been done — the resulting model is very simple, and the conditions concise and intuitive [26].

Following a discussion of the specification and synthesis methods, illustrations are given in [29] that demonstrate the use of the proposed theoretical basis in synthesizing various classes of algorithms. It is shown how (families of) systolic algorithms may be obtained as a special case. Methods for proving the correctness of implementations are presented and illustrated with examples. The concept of the propagation of computational loci arises naturally in course of the development, and serves to generalize the commonly used notion of a "wavefront" of computation for 2–dimensional architectures. Automatable design aids based on the proposed algebraic basis are delineated. Finally, it is shown how MOS circuits can be

modelled using the primitives available, and the algebraic derivation of Bryant's simulation algorithm used in MOSSIM II is illustrated in this context.

3.2.1. Interface With Diana

Most of our transformation tools use the parse tree representation of a program as the primary data structure they work with. We have in mind the long term objective of being able to interface with the tools that are designed to operate on Ada program parse trees, and that being developed by the Ada community at large (and in particular the DARPA community). To this end, we have been interacting (to a limited extent) with the Diana group (primarily at Tartan Laboratories).

3.3. Some Remarks on System Implementation Issues

While we are continuing work on the current version of the transformation system (in Interlisp, and on the Vax and DEC-20), it has become clear that there are two major deficiencies that need to be remedied sooner or later. These are (1) unsuitability of the current parse tree interface (and parser generator) for several of the transformation routines themselves; and (2) (lack of) speed: this is due to the slowness of Interlisp on the Vax (compounded, of course, by the fact that we are working with non-trivial pieces of software).

To solve the first problem, it is necessary to redesign the parser generator (which has been imported from ISI [31]). However, since the other tools (particularly the syntax directed editor generator and pattern matching system) and the history list mechanism are all very much inter-related and quite deeply ingrained in the system, there is a substantial software development effort involved in doing this. Currently, we have neither the equipment nor the man-power to support such an effort. We envision the redesign being more profitably done using a newer generation of Lisp (e.g. PSL, CommonLisp) for efficiency reasons, and run on personal machines, rather than on a Vax like machine. In the interim, however, the response of the extant version of our system can also benefit greatly from being run on an Interlisp-supporting machine, e.g., the Dorado/Dolphin. Having access to such systems would obviously result in greatly improved programmer productivity.

4. PPL Design Activities

by

Kent F. Smith, Brent Nelson, Tony Carter, and Alan Hayes

A system for the design of integrated circuits using a methodology known as Path Programmable Logic (PPL) has been developed by the Utah VLSI Group. This work has been sponsored in part by the DARPA contract and by contracts with other government agencies and in part by support from several independent companies. The system addresses the complete design cycle including initial logic design, circuit layout, simulation, electrical checking, and pattern generator tape preparation. It includes: (1) symbolic layout programs to facilitate the placement of the symbols on the grid, (2) a simulator patterned after switch-level simulators but specifically tailored for use on PPL, (3) a checker program for cell placement verification and DC circuit loading checking, and (4) a common database for design representation.

4.1. PPL Design Characteristics

The characteristics of design using the PPL methodology include:

1. IC design is performed by placing small circuit modules which can be represented with logic symbols on a grid representing the integrated circuit. When the grid is completely populated, it is both the logical representation and the topological layout of the circuit. Efficient design changes can be made as a result of this design methodology because the designer has simultaneous perception of the circuit function and the circuit topology.
2. The circuit modules have predefined schematic and composite representations. They are custom designed to optimize performance and size for any specific integrated circuit process. Design Rule Checking (DRC) is performed on the module and thus it is not necessary to do DRC on the overall circuit since it is simply a collection of circuit modules.
3. A complete circuit can be designed in PPL and no custom design is required. The pads and the interconnect can also be made by the placement of PPL cells on the grid. All interconnections between modules are there by default. The designer only places breaks to remove connections rather than to add them.
4. Hierarchical design is possible by custom design of macros which are collections of PPL cells put together to perform specified functions. These macros cells can have custom physical shapes to conform to specific space requirements.
5. Simulation and checking are easily accomplished, eliminating the need for very difficult and time-consuming operations. The only elements manipulated are symbols rather than transistors or rectangles which must be checked in systems that design at the transistor level.

4.2. The Analogy Between the PPL Design and a Computer Program

There is an analogy between the development of the PPL design methodology and programming languages. The 1's and 0's which were used in early machine language computer programming are analogous to the rectangles which are used in the custom layout of integrated circuits. Placing transistors on a composite might be thought of as being analogous to writing machine language code in hexidecimal since we are still placing rectangles on a grid in shorthand form. The PPL design methodology is analogous to writing programs in assembly language where mnemonics are used to represent specific collections of transistors (functions). This PPL design methodology is still very dependent upon the specific technology which it is designed in. This is similar to the way that assembly language is machine-dependent.

The analogy between the development of computer programs and the PPL methodology can be carried even further with the compilation of high level circuit description languages to

integrated circuit layouts (silicon compilers). The high level descriptions of the integrated circuit are machine independent and are compiled directly to a specific PPL cell set designed in a particular technology. To date there have been cell sets done in NMOS [21], CMOS [22], and I2L [23]. An example of such a silicon compiler is ASSASSIN [7] which is currently in use at the University of Utah.

4.3. Design Time vs. Integrated Circuit Area

The main disadvantage of PPL design methodology is that it will probably result in circuits which are larger than completely custom-designed circuits. Previous work done by the VLSI group at the University of Utah has compared some custom designs to some PPL designs. This gives insight into the tradeoffs which exist between the two techniques. A circuit known as the Utah Serial Cordic Machine (USCM) was designed under a contract with Wright Patterson AFB for the VHSIC program [3, 4, 5] using both custom design techniques and the PPL Design Methodology. The USCM was constructed using an implementation similar to the shift-register scheme proposed by Volder [30].

The USCM was implemented using a CMOS PPL cell set. Its design time and chip area were compared to those for an equivalent custom NMOS design done at Boeing Aerospace Corp. The entire CMOS PPL chip was designed and simulated in approximately eight man days, compared to approximately eighty man days for the NMOS custom design. The CMOS PPL design was 19 percent larger than the custom NMOS design. While these figures may not be an accurate reflection of the variables which enter into design time measurements, they are indicators that PPL designs require significantly less design time than do equivalent custom designs and result in chips which are not significantly larger in area.

This favorable reduction in design time can be attributed to several factors: (1) The designer has concurrent perception of logical function and layout. Thus, he can immediately see when the logic function being implemented does not fit in well with the rest of the circuit. The logic design is made as the composite is drawn. This eliminates the need for separate composite layout/logic design stages. (2) The higher level symbolic notation allows the designer to manipulate very complex logical elements in an efficient manner. It is, for example, not necessary to trace a complex series of logic gates to determine the function of the circuit because the symbolic notation is easily read and interpreted. In addition, the symbolic notation can be directly simulated and does not require the extraction of the transistor-level circuit from the composite.

Past experience would indicate that the area penalty incurred by the PPL design methodology will eventually disappear as more sophsticated design tools are developed. This is again analogous to the development of compilers. It is well known that, as expertise in compiler writing improved, the gap between hand-coded and compiler-produced object code size became negligible. Some of the techniques being developed for compaction of integrated circuit layouts will be used to close the current gap between the area required for custom designs and automatically generated PPL layouts.

4.4. The Utah PPL Design System

In addition to the development of the PPL as a hardware implementation methodology described above, the other major thrust of research here at Utah has been in developing software tools for PPL design. The goals of this software research have included the following: (1) Finding ways to exploit the symbolic nature and representation of a PPL design to reduce design complexity. (2) Development of CAD tools around conventional computer hardware, which would allow designers to work from remote workstations. (3) Creation of a complete system to be used by the IC design community here at Utah.

An integral part of the design system is a Computer Vision CADDS2/VLSI Designer System. It is used to do the composite layout of the individual PPL cells, placement of the individual cells on a grid to form a circuit, connecting the circuit to pads, adding scribe lanes, and generating a PG tape. Although we have relied heavily on this machine in the initial development of the system, in its absence all of the functions it performs could be done with other tools (the Cal-Tech Software Package for example).

The other part of the design system is built around a DECSystem-20. A silicon compiler for finite state machines (FSM), a symbolic layout system, a simulator and cell placement checker, and a compaction program all reside there. The transfer of designs between the Computer Vision machine (CV) and the DECSystem-20 is done using a mag tape written in Computer Vision External Database format. The combination of these two computers gives the system the power of the CV's IC layout features combined with the computing power of a mainframe.

Each PPL cell used in the system has three representations. The composites of the cells are designed so that they fit together by virture of their being placed adjacent to each other on the grid. A schematic representation of each cell is created for reference. A graphical representation is also created which is used by the designer as he uses the cells to form larger circuits.

4.5. Presently Existing Circuit Layout Tools

The placement of the PPL cells on the grid to form a circuit can be done using either the Computer Vision machine or one of several programs on the Utah DECSystem-20. The program used for cell placement on the DECSystem-20 is known as SLED (Structured Logic Editor) [15]. In SLED, the PPL design is represented as an array of cell symbols which are then edited. With the SLED editor, a simple CRT terminal and modem is all that is needed for circuit design but at the expense of more cryptic graphical representations of the individual PPL cells than those found on the Computer Vision machine. In general, the ability to use SLED from a remote terminal outweighs this limitation. Advanced editors are now being designed to run on a CRT terminal that will overcome some of the graphical limitations of SLED.

SLED was designed to be similar to a screen-oriented text editor. In fact, the commands in SLED are the same as the equivalent commands in EMACS [8], a popular screen-oriented text editor. Cursor movement is possible in any of the four directions, and regions (windows) can be marked and then named, deleted, replicated, or written to a disk file. Conventional text editors, however, only allow for scrolling and windowing in the vertical direction (lines longer than the width of the screen are wrapped around). In SLED, scrolling and windowing are possible in both directions. Thus, an array with 300 columns and 300 rows can be displayed and edited using SLED without screen wrap-around. The effect is that the user has an 80X24 window which can be moved around the array.

Circuit layout can also be accomplished using a first-generation silicon compiler. Compilaton of Ada language modules to circuits is accomplished using the program named ASSASSIN [7]. This program takes as its input a textual description of the operation of a control unit (Finite State Machine) and from it generates a PPL layout implementing the control unit.

4.6. Circuit Simulation and Electrical Checking

Simulation of the PPL design is essential before actual fabrication. An important part of the design system is a simulator (ASYLIM) which can do simulation of the PPL. Because the PPL cells are simulated and checked individually at the transient level when the cell set is designed, the complete circuit made up of PPL cells can be simulated at a switch or gate level. ASYLIM [16, 17] reads the circuit database written in Computer Vision External Database format. Thus, the actual design can be simulated rather than a logic equivalent.

ASYLIM is similar to other recently developed MOS simulators in that it uses a switch model. However, the development of a simulator for PPL has shown [17] that a special purpose simulator was required in order to preserve the user's abstract view of the circuit. The input format to existing simulators is typically given in the form of a table or listing of transistors and nodes. To preserve the user's abstract view of the circuit it was necessary to design a simulator for PPL where the elements in the simulator correspond to those in the PPL cell set. During the interactive debugging phase of the simulation of a circuit, the user can then refer to circuit elements by their **position** in the PPL array. An added feature of the PPL simulator is that the information stored in the simulator's internal representation of the

circuit interconnect structure can be used for additional circuit checking unique to the PPL methodology. The end result is that ASYLIM is similar to conventional switch-level simulators but with an extensive user-interface that allows the user to work with the circuit at the symbolic PPL level, the same level he uses when designing.

ASYLIM makes use of six-valued logic and uses a unit-delay timing model [1, 2]. The underlying circuit model primitives are switches but with extensions to allow for the simulation of certain entities as gates (flip flops and latches). It has been shown that the unit-delay model is adequate provided the circuit is free from races. Thus it can be used to model the sequence of circuit activity [2].

An additional advantage of using ASYLIM over other simulators is that it contains an extensive interactive circuit debugger. The features of this debugger allow the user to view the circuit interconnect structure as constructed by the simulator. This is displayed in a readable format that allows the user to quickly compare the simulator's interpretation of the circuit element interconnections and the intended design. This comparison uncovers most design errors relatively quickly. In addition, the simulator performs a pre-simulation plausibility check on the circuit's nodal structure. This feature (the idea borrowed from Bryant's MOSSIM [2] enables the user to find a large percentage of the design errors without ever going to the expense of an actual simulation. This check identifies nodes with fanout but no inputs, inputs but no fanout, no path to either power or ground, or multiple pullup loads.

While a logic or switch-level simulation can provide an invaluable service in verifying the logic design, there are many features of a design that do not show up in a simulation run. For example, the ground node may be specified as an input to a transistor in a diagram but it requires an explicit check on the layout to ensure that ground actually has been routed to that device. In PPL design, these types of electrical (non-logic) entities are included in the design using special cells. For instance, the power bussing structure is included by placing power and ground buss cells around the circuit perimeter. In addition, other cells, like row and column loads, are usually left of out of logic diagrams but must be included for the circuit's correct operation.. ASYLIM checks for these cells as a part of its operation.

4.7. Self Timed IC Design with PPL's

Another activity which has been funded by a private company and is of importance in the development of the PPL methodology is the design of self-timed modules using the PPL cell set. The work is based on techniques developed earlier [9] for realizing self-timed stored state sequential circuits. The original investigations were applied to off-the-shelf SSI parts. The present investigations are for the transfer of those ideas to large collections (macros) of PPL cells for use in the design of self timed systems to be contained on single integrated circuits. The investigations have led to further development of the PPL cell set to include methods for self timed circuits [11].

This research has resulted in a design discipline for self-timed stored state machines which has been developed using a conventional single rail Four Cycle signalling protocol. (State descriptions are encoded in PLAs represented in PPL.) The discipline differs from that used by Carter [7] which uses a technique known as a "one hot" scheme. The approach used for realizing the self timed stored state machines is based on two key developments: (1) A novel clocking circuit that generates a non-overlapping two phase clock cycle for an arbitrary size register, where the duration of the phi 1 phase of the cycle is automatically adjusted to the register size, and (2) A layout discipline for the folded PLA holding the state table, which guarantees that the inputs to the state register will be valid at the time that the clock cycle occurs.

The method depends on certain properties of the NMOS PPL cell set, i.e. that row and clock wires are polysilicon, and that registers are formed by locating flip-flop cells such that their clock lines are serially connected. This method offers a designer the advantage that he need not concern himself with the timing details of a state machine design in order to assure that it will work. Assuming that the state table realized by the PLA is correct, that the rows and columns of the design are properly loaded, and that the proper interconnections have been made (all of which can be verified with the PPL simulator [17]), the designer can be assured of correct operation of the state machine. The principle disadvantage of the method is the

overhead of the clocking circuit which must be associated with each state machine.

In addition to the self–timed state machine design, the described design discipline [11] has been applied to several interesting types of self–timed data–path modules, for example multi–bit latches and ripple–carry counters.

4.8. Future CAD Tools for the PPL Design Methodology

Our operational design tools should be enhanced. The following agenda lists the tools we have identified as being an important part of a design system for this methodology and which we plan to develop:

1. A Relational PPL Database Management System — This will allow the same software tools such as the editor and simulator to be used on PPL designs done using any specified integrated circuit technology such as NMOS, CMOS, I2L, and GaAs. In addition, it will provide a standard interface between the various CAD programs.
2. A Symbolic, Interactive, PPL Editor — this editor will be used to create a symbolic representation of a PPL circuit. It will be used interactively by a designer for the semi–automatic placing of PPL cells on the PPL grid. Because of the symbolic nature of PPL, many of the mundane design tasks can be automatically performed by the editor, leaving the designer free to concentrate on logical design. The editor will use either tablet or keyboard entry with simultaneous graphical representation of both the logic description and the circuit topology.
3. Minimization of PPL programs — Development of a compaction program for compressing a PPL design by rearranging its symbolic description. Such a program will use heuristically driven artificial intelligence techniques to arrive at a near–optimal solution to the minimization problem. This tool will give us the capability of doing loosely packed PPL designs which can then be automatically compressed. This is a unique feature of the PPL design methodology and can be accomplished because of the symbolic nature of the PPL.
4. Predefined Structured Logic Blocks — We are persuaded that circuits that already contain large blocks of non–PPL structured logic should be designed using similar techniques to those presently used for the design of such blocks. For instance, if a random access memory (RAM) is required in a circuit, it is more efficient, both from a performance as well as a topological standpoint, to actually do a custom layout of the RAM. The PPL cell set can be extended to include very elementary cells from which macro cells can be developed for any specific implementation of a RAM. Components generated by such an implementation, although not strictly PPLs, would be compatible with their PPL neighbors. A list of of structures we expect to implement as macros includes:

```
nxm ram
nxm rom
n-bit ripple adder
n bit fast adder
n-bit priority encoder
n-bit register
nxm multiplier
n-bit comparator
n-bit synch counter
n-bit ripple counter
n-bit by m:1 MUX
```

4.9. Observations

Our research thus far has demonstrated the usefulness of the PPL methodology as a higher level design technique for hardware analogous to the use of assembly language for computer programming. The analogy has been extended by the introduction of ASSASSIN, a first–generation silicon compiler for speed–independent finite state machines.

Our design system has proven useful for doing actual design of a variety of integrated circuits. It has reduced design times required by an order of magnitude. Resultant designs are easily simulated and corrected due to their symbolic representation. System designers with little or no direct experience with integrated circuit design can do actual IC layout.

5. Project Bibliography of Papers, Reports and Theses

This section contains a cumulative list of the papers, reports and theses regarded as direct or indirect "products" of this Project. Subsequent semiannual technical reports will contain updated versions of the list given here.

[1] Carter, T.M.
ASSASSIN: An Assembly, Specification and Analysis System for Speed-Independent Control-Unit Design in Integrated Circuits Using PPL.
Master's thesis, University of Utah, Department of Computer Science, June, 1982.

[2] Carter, T.M.
ASSASSIN: A CAD System for Self-Timed Control-Unit Design.
Technical Report UTEC-82-101, University of Utah, October, 1982.

[3] Drenan, L.A.
On Transforming Ada to Silicon.
Master's thesis, University of Utah, Department of Computer Science, August, 1982.

[4] Drenan, L.A., Organick, E.I.
Ada to Silicon Trnsformations: The Outline of a Method.
Technical Report UTEC-82-016, University of Utah, Dept. of Computer Science, Sept, 1982.

[5] Hayes, A.B.
Self-Timed IC Designs with PPL's.
October, 1982.
Paper submitted for 1983 Cal Tech VLSI Conference.

[6] Nelson, B.E.
ASYLIM User's Manual
1982.

[7] Nelson, B.E.
ASYLIM: A Simulation and Placement Checking System for Path-Programmable Logic Integrated Circuits.
Master's thesis, University of Utah, Department of Computer Science, October, 1982.

[8] Organick, E.I., and Lindstrom, G.
Mapping high-order language units into VLSI structures.
In *Proc. COMPCON 82*, pages 15-18. IEEE, Feb., 1982.

[9] Organick, E.I., Carter, T., Lindstrom, G., Smith, K. F., Subrahmanyam, P.A.
Transformation of Ada Programs into Silicon. SemiAnnual Technical Report.
Technical Report UTEC-82-020, University of Utah, March, 1982.

[10] Organick, E.I., Carter, T.M., Hayes, A.B., Nelson, B.E., Lindstrom, G., Smith, K., Subrahmanyam, P.A.
Transformation of Ada Programs into Silicon. Second SemiAnnual Technical Report (to appear).
Technical Report UTEC-82-103, University of Utah, November, 1982.

[11] Ramachandran, R.
A Complexity Computation Package for Data Type Implementations.
Master's thesis, University of Utah, Department of Computer Science, June, 1982.

[12] Subrahmanyam, P.A.
From Anna+ to Ada: Automating the Synthesis of Ada Package and Task Bodies.
Technical Report Internal Report, University of Utah, March, 1982.

[13] Purushothaman.S, and Subrahmanyam, P.A.
An Algebraic Model of Seitz's Weak Conditions for Self Timed Systems.
Technical Report UTEC # 82-066, University of Utah, October, 1982.

[14] Subrahmanyam, P.A.
Language Issues in Transformation Systems (to appear).
Technical Report UTEC # 82–069, University of Utah, November, 1982.

[15] Subrahmanyam, P.A. and Rajopadhye, S.
Automated Design of VLSI Architectures: Some Preliminary Explorations.
Technical Report UTEC # 82–067, University of Utah, October (Revised), 1982.

[16] Subrahmanyam, P.A.
A Theoretical Basis for the Synthesis and Verification of Systolic Designs.
Technical Report Internal Report, Dept. of Computer Science, University of Utah, June, 1982.

[17] Subrahmanyam, P.A.
On Automating the Computation of Approximate, Concrete, and Asymptotic Complexity Measures of VLSI Designs (to appear).
Technical Report UTEC–82–095, Dept. of Computer Science, University of Utah, November, 1982.

[18] Subrahmanyam, P.A.
Automatable Paradigms for Software–Hardware Design: Language Issues.
In J.Rader (editor), *IEEE Workshop on VLSI and Software Engineering*. IEEE, October, 1982.
Also available as University of Utah Technical Report UTEC–82–096, September 1982.

[19] P.A. Subrahmanyam.
An Automatic/Interactive Software Development System: Formal Basis and Design.
North–Holland, Amsterdam, 1982, .

[20] Subrahmanyam, P.A.
Abstractions to Silicon: A New Design Paradigm for Special Purpose VLSI Systems.
Technical Report UTEC # 82–065, University of Utah, January, 1981 (Revised May 1982).
Submitted for Publication to TOCS.

[21] Subrahmanyam, P.A.
An Algebraic Basis for VLSI Design.
Draft of a Research Monograph, April 1982, 120 pp. Available from the Department of Computer Science, University of Utah.

6. Appendix

```
-------------------------------------------------------------------------
--                                                                     --
--                        Rda-to-Silicon Project                       --
--                          University of Utah:                        --
--                                                                     --
--                DoD Internet Protocol INM_OUT submodule              --
--                                                                     --
--          Ada code for the body of task Read_Init_Parameters         --
--                     Version of October 25, 1982                     --
-------------------------------------------------------------------------

separate (Inm_Out_Module)


task body Read_Init_Parameters is

  -- Accessed globals:
  --------------------
  -- number_of_local_net_types_of_service:          octet_type
  -- local_net_type_of_service_table_row_size:      octet_type
  -- tos_table:                                     octet_buffer_type

  -- Renamed task entry:
  ----------------------
     -- The package Memory_Module containing the task Memory holds
     -- to-be-sent datagrams as well as initialization parameters
     -- needed by INM_OUT.

  procedure Memory_request(
      request_type_formal:          memory_request_type;
                                       -- Load_address or receive_datum_octet.
      chunk_of_address_formal:      chunk_of_address_type;
                                       -- Don't care when request_type_formal
                                       -- receive_datum_octet.
      octet_formal:              out octet_type)
                                       -- Don't care when load_address.
    renames Memory.Request;

  -- Local variable declaration:
  ------------------------------
  -- The following variable is commented out. It appeared only in the
  -- "high-level" used to read in the TOS table.  See below.
  --  number_of_tos_table_octets: integer range 2 .. max_tos_table_size - 1;
  octet_register:                 octet_type;

begin
  loop
    accept Go(
        init_num_formal:            bit4;                -- For Carter's paper
                                                         -- only; otherwise bit3
        response:             out out_response)
      do
        response := sent_ok;                              -- Also means init_ok.

        -- Get from the server all of the addr_chunks needed to form the base
        -- address in memory that holds the initialization parameters and
        -- sends these chunks to the Memory module.
        for index in 1 .. init_num_formal
        loop
          accept Srv_req(                                 -- Get next address
                                                          -- chunk from the
                                                          -- Server Module.
              server_command_datum:     srv_command;
              response_to_server:   out out_response)
          do
            Memory_request(                               -- Put chunk out to the
                                                          -- Memory module.
```

```
            request_type_formal      => load_address,
            chunk_of_address_formal  =>
                                    Convert_srv_command_to_chunk_of_address
                                        (server_command_datum),
            octet_formal             => dont_care_octet);
      end Srv_req;
    end loop;

    -- Get the 6 individual initialization parameters (contained in the
    -- next 8 octets received) from the Memory Module.
    for index in 1 .. 8
    loop

      Memory_request(
          request_type_formal      => receive_datum_octet,
          chunk_of_address_formal  => dont_care_X_datum,
          octet_formal             => octet_register);

      case index is
        when 1 => inm_max_packet.lo           := octet_register;
        when 2 => inm_max_packet.hi           := octet_register;
        when 3 => inm_address_length          := octet_register;
        when 4 => inm_time_out.lo             := octet_register;
        when 5 => inm_time_out.hi             := octet_register;
        when 6 => ack_type                    := octet_register;
        when 7 => local_net_type_of_service_table_row_size
                                              := octet_register;
        when 8 => number_of_local_net_types_of_service
                                              := octet_register;
      end case;
    end loop;

    -- Convert the local net timeout into milliseconds.?
    -- time_out_in_milliseconds := inm_time_out / 1000.0;
                                      -- Left-hand side variable declared
                                      -- in Inm_Out_Module. Value is used
                                      -- later in Do_send procedure.
                                      -- Note: Davis never did this in
                                      -- his design. Is this step needed?
                                      -- No! We don't need this step
                                      -- since the  quotient can be
                                      -- approximated by a div by 2**10
                                      -- in the event we need to
                                      -- represent milliseconds.

    -- Read in type of service translation table.

      --    The following code in comments is replaced below by a
      --    "lower-level" version that closely reflects the hardware
      --    implementation chosen in which we eliminate the need for
      --    for a multiplier.

--    number_of_tos_table_octets := local_net_type_of_service_table_row_size
--                                * number_of_local_net_types_of_service;

--    -- Check to see if required table size exceeds maximum
--    if  number_of_tos_table_octets > max_tos_table_size  then
--      response := bad_srv_command;
--      return;
--    end if;

--    for index in 1 .. number_of_tos_table_octets
--    loop

--      Memory_request(
--          request_type_formal      => receive_datum_octet,
--          chunk_of_address_formal  => dont_care_X_datum,
--          octet_formal             => tos_table(index));
--    end loop;
```

```
        declare
          row_number: integer range 0 .. number_of_local_net_types_of service;
          col_number: integer range 0 ..
                                  local_net_type_of_service_table_row_size;
          index:      integer range 0 ..
                                  number_of_local_net_types_of service
                                  * local_net_type_of_service_table_row_size
                                  := 0;
        begin
          row_number := 0;
          loop                        -- Outer loop reads all rows of TOS table.
            col_number := 0;
            loop                      -- Inner loop reads in one row of TOS table.
              Memory_request(
                  request_type_formal      => receive_datum_octet,
                  chunk_of_address_formal  => dont_care_X_datum,
                  octet_formal             => tos_table(index));

              col_number := col_number + 1;
              exit when col_number = local_net_type_of_service_table_row_size;

              index := index + 1;
              if index > max_tos_table_size then
                response := bad_srv_command;
                return;               -- Exit the current accept statement.
              end if;
            end loop;                 -- End inner loop.

            row_number := row_number + 1;
            exit when row_number = number_of_local_net_types_of_service;
          end loop;                   -- End outer loop.
        end;                          -- End declare block.

      end Go;                         -- End of init processing.

    end loop;                         -- End of outer-most (inifinite)
                                      -- loop.
end Read_Init_Parameters;
```

References

[1] R. E. Bryant.
Logic Simulation of MOS LSI.
PhD Disseration Proposal, Massachusetts Institute of Technology, January, 1980.

[2] R. E. Bryant.
A Switch-Level Simulation Model for Integrated Logic Circuits.
PhD thesis, Massachusetts Institute of Technology, 1981.

[3] T. M. Carter and K. F. Smith.
Applications of Logic Arrays in VHSIC Design.
March, 1981.
Quarterly Technical Report #2 from the VLSI Research Group at the University of Utah, Department of Computer Science, to Boeing Aerospace Company.

[4] T. M. Carter; K. F. Smith; C. E. Hunt; and W. L. Howard.
Applications of Logic Arrays in VHSIC Design.
June, 1981.
Quarterly Technical Report #3 from the VLSI Research Group at the University of Utah, Department of Computer Science, to Boeing Aerospace Corporation.

[5] T. M. Carter; K. F. Smith; C. E. Hunt; and B. E. Nelson.
Applications of Logic Arrays in VHSIC Design.
September, 1981.
Quarterly Technical Report #4 from the VLSI Research Group at the University of Utah, Department of Computer Science, to Boeing Aerospace Corporation.

[6] Carter, T.M.
ASSASSIN: A CAD System for Self-Timed Control-Unit Design.
Technical Report UTEC-82-101, University of Utah, October, 1982.

[7] T. M. Carter.
ASSASSIN: An Assembly, Specification and Analysis System for Speed-Independent Control-Unit Design in Integrated Circuits Using PPL.
Master's thesis, Department of Computer Science, University of Utah, June, 1982.

[8] Richard M. Stallman.
EMACS Manual for TWENEX Users
Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1980.

[9] A. B. Hayes.
Stored State Asynchronous Sequential Circuits.
IEEE Transactions on Computers C-30(8):596-600, August, 1981.

[10] Alan B. Hayes.
High-level Logic Design of the DoD INM-OUT Module.
April, 1982.

[11] Hayes, A.B.
Self-Timed IC Designs with PPL's.
October, 1982.
Paper submitted for 1983 Cal Tech VLSI Conference.

[12] Krieg-Bruckner, B., Luckham, D.C., von Henke, F.W., Owe, O.
(Draft) Reference Manual for Anna, A language for Annotating Ada Programs.
Unpublished, Reviewer's Copy, October 1982.

[13] Lindstrom, G.
Internet Protocol Case Study: Background and Initial Design.
May, 1982.

[14] Lindstrom, G., Organick, E.I., Klass, D., Maloney, M.
Ada Specifications for the DoD Internet Protocol: The INM_OUT Submodule, Report No. 1.
Technical Report, Department of Computer Science, University of Utah, November, 1982.

[15] Brent E. Nelson.
SLED User's Manual
1982.
Department of Computer Science, University of Utah.

[16] Brent E. Nelson.
ASYLIM User's Manual
1982.
Department of Computer Science, University of Utah.

[17] Brent E. Nelson.
ASYLIM: A Simulation and Placement Checking System for Path-Programmable Logic Integrated Circuits.
Master's thesis, University of Utah, October, 1982.

[18] Organick, E. I., and Lindstrom, G.
Mapping high-order language units into VLSI structures.
In *Proc. COMPCON 82*, pages 15-18. IEEE, Feb., 1982.

[19] Organick, E.I., Carter, T.M., Lindstrom, G., Smith, K.F., Subrahmanyam, P.A.
Transformation of Ada Programs into Silicon. SemiAnnual Technical Report.
Technical Report UTEC-82-020, University of Utah, March, 1982.

[20] Postel, Jon: editor.
Internet Protocol: DARPA Internet Program, Protocol Specification.
Technical Report RFC 791, Information Sciences Institute, USC, Sept., 1981.

[21] K. F. Smith.
Implementation of SLA's in NMOS Technology.
In *Proceedings of the VLSI 81 International Conference, Edinburgh, UK*, pages 247-256. August, 1981.

[22] K. F. Smith; T. M. Carter; and C. E. Hunt.
The CMOS SLA and SLA Program Structures.
In H. T. Kung; B. Sproull; and G. Steele (editor), *Proceedings of the 1981 CMU Conference on VLSI Systems and Computatons*, pages 396-407. Computer Science Department, Carnegie-Mellon University, Computer Science Press, October, 1981.

[23] K. F. Smith.
Design of Stored Logic Arrays in I2L.
In *Proceedings of the 1981 IEEE International Symposium on Circuits and Systems*, pages 105-110. IEEE Circuits and Systems Society, April, 1981.
IEEE Catalog No. 81CH1635-2.

[24] Subrahmanyam, P.A. and Rajopadhye, S.
Automated Design of VLSI Architectures: Some Preliminary Explorations.
Technical Report UTEC # 82-067, University of Utah, October (Revised), 1982.

[25] Subrahmanyam, P.A.
From Anna+ to Ada: Automating the Synthesis of Ada Package and Task Bodies.
Technical Report Internal Report, University of Utah, March, 1982.

[26] Purushothaman.S, and Subrahmanyam, P.A.
Algebraic Modeling of Self Timed Systems.
Technical Report UTEC # 82-066, University of Utah, August, 1982.

[27] Subrahmanyam, P.A.
A Theoretical Basis for the Synthesis and Verification of Systolic Designs.
Technical Report UTEC–82–097, Dept. of Computer Science, University of Utah, June, 1982.

[28] Subrahmanyam, P.A.
Transformational Implementation of Software/Hardware Systems: Global Strategy Guidance.
Submitted for Publication, University of Utah, January, 1982.

[29] Subrahmanyam, P.A.
An Algebraic Basis for VLSI Design.
Draft of a Research Monograph, April 1982. Available from the Department of Computer Science, University of Utah.

[30] J. E. Volder.
The CORDIC Trigonometric Computing Technique.
IRE Transactions on Electronic Computers Volumn Number Unknown:330= 334, September, 1959.

[31] Wile, Dave.
POPART: A Producer of Parsers and Related Tools, System Builder's Manual.
June 1980.
Unpublished, USC/ISI.