Automatic Rapid Prototyping of Semi-Custom VLSI Circuits using FPGAs

Jae-tack Yoo, Kent F. Smith, Erik Brunvand

UUCS-94-022

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

August 24, 1994

Abstract

We describe a technique for translating semi-custom VLSI circuits automatically, integrating two design environments, into field programmable gate arrays (FPGAs) for rapid and inexpensive prototyping. The VLSI circuits are designed using a cell-matrix based environment that produces chips with density comparable to full custom VLSI design. These circuits are translated automatically into FPGAs for testing and system development. A four-bit pipelined array multiplier is used as an example of this translation. The multiplier is implemented in CMOS in both synchronous and asynchronous pipelined versions, and translated into Actel FPGAs both automatically, and by hand for comparison. The six test chips were all found to be fully functional, and the translation efficiency in terms of chip speed and area is shown. This result demonstrates the potential of this approach to system development.

1 Introduction

Recent developments in circuit technology have resulted in a wide variety of choices for building application specific digital hardware systems. The design space ranges from custom and semi-custom integrated circuit design, to mask-programmed gate arrays, to field programmable devices that can be programmed in minutes. Each of these technologies, of course, has their own set of advantages and drawbacks. Custom integrated circuits have the most flexibility, and the highest circuit density of any of these choices, but require a relatively long design and fabrication cycle, and high cost if only a small number of chips are needed. Mask programmable gate arrays offer reduced design and fabrication time with some reduction in circuit performance and density. Field programmable gate arrays (FPGAs) are gate arrays with programmable links between the logic modules. These devices can be programmed in minutes and offer the fastest turnaround time of any of the technologies, and the lowest cost per unit in small quantities. The drawbacks are again in circuit density and speed.

In this paper we describe a technique that allows custom[1] VLSI circuits to be prototyped quickly and inexpensively using FPGAs. Ideally, the designer would like to try out a design using inexpensive and quickly programmed devices like FPGAs, and when the design is working correctly, recast that design into a faster technology such as custom CMOS. Unfortunately, this recasting and remapping to a custom technology is typically not easy and it can take a great deal of time to redesign the original FPGA circuit. Our method involves designing the custom circuit first using a cell-matrix based design tool, and then translating that custom design automatically into an FPGA for testing. If the FPGA circuit is correct, no additional work is needed to rebuild the design in a faster denser technology because that was the original starting point. If the design is not satisfactory, changes are made to the custom circuit, and a new FPGA circuit is generated for testing. If there is an urgent need for the circuit, the FPGA version may be used while the CMOS chip is being fabricated. In particular, we use the Physical Placement of Logic (PPL) design system [1, 2] to design the custom CMOS circuits, and map those designs automatically into Actel FPGAs [3] for prototyping and testing.

Our goal is to exploit the advantages of both the PPL and Actel environments by developing an automatic circuit translator which will translate a PPL circuit into an Actel FPGA circuit. PPL circuits can be tested by simulation as before, or by translating to an Actel FPGA and testing the Actel chip. In this paper we describe an automatic translator based on gate-by-gate/cell-by-cell translation. We also show, through a hand-translated example, extensions to this prototype translator that improve the efficiency of the translated circuit. The translation procedure is illustrated using a 4-bit pipelined multiplier designed in PPL in both synchronous and asynchronous versions. This multiplier is translated into an Actel circuit automatically, and by hand as a comparison. Two MOSIS chips were fabricated and four Actel chips were programmed. All the chips are functionally correct, and performance figures for each chip are presented.

[1]Because the term “semi-custom” is somewhat awkward, and ill-defined, we will use the term “custom VLSI” to encompass both custom and semi-custom approaches to VLSI design

2 Circuit Technologies

The goal of this translation procedure is to convert PPL custom VLSI designs automatically into equivalent Actel FPGA circuits. The input to the translator is a netlist generated by the PPL tools, and the output is a new netlist which is used by CAD tools from Viewlogic and Actel to produce Actel FPGA circuits.

2.1 PPL Design Environment

PPL is a cell-based physical VLSI design environment which includes graphical placement tools and a variety of simulation models for the resulting circuits. This has proven to be an effective design tool for complex VLSI circuits and can produce circuits whose density and speed are comparable to that of full custom design [1, 2]. This cell-based design paradigm is constrained through the use of a fixed layout grid for cell placement and routing, and uses a set of predefined layout descriptions, similar to standard cells but at a finer grain and with two-dimensional interconnection.

Because PPL circuits are custom VLSI circuits, the fabrication time is around 8 weeks (through the MOSIS facility). This fabrication time is long with respect to the design cycle for the chip, and means that prototyping and testing must be done using simulation. In-circuit testing must wait for chips to be returned.

The PPL environment includes a wide variety of tools for the design and evaluation of custom integrated circuits. Circuits are designed using a graphical editor called *ACME*, or a character based editor called *tiler*. Simulation is accomplished using a switch-level simulator called *simppl*, or by simulating pieces of the circuit at the transistor level using *hspice*. The PPL environment also contains a circuit extractor and electrical rules checker called *simpplex*, and a netlist generator called *spplice* that operate on the circuit database [4]. Our translator interfaces to the PPL environment by modifying the *spplice* netlist generator to translate the circuit from a set of PPL cells to an equivalent set of Actel cells.

2.2 Actel Design Environment

In contrast to the custom circuits designed using PPL, Actel FPGAs can be "fabricated" in minutes. The Actel product is a field programmable gate array (FPGA) implemented in 2.0μ or 1.2μ CMOS. The chip is arranged much like a conventional channeled gate array with rows of logic cells interspersed with routing channels. However, these routing channels are not simply empty space where the wire may be run, as in a mask programmable gate array. The routing channels on an Actel chip contain predefined wire segments of various lengths. These wire segments may be connected through a two-terminal electrically programmable device known as an "anti-fuse." An anti-fuse is a device that changes irreversibly from a high to a low resistance when programmed by applying an appropriate voltage across its terminals. Logic modules are connected to perform the desired function by programming selected anti-fuses and thus programming the chip to a specific function. Once programmed, an Actel FPGA cannot be changed or re-programmed.

Table 1: Actel ACT-1 Chip Sizes

Device	Logic Modules	Package	User I/Os
A1010	295	68 PLCC	57
A1020	547	84 PLCC	69

The Actel ACT-1 architecture is chosen as the target for our translation. The basic ACT-1 logic module is an eight input, single output, device chosen to be small and flexible and at the same time be a good match for the wiring resources available on the chip. The basic module is shown in Figure 1. This module is the basis for all the macros in the Actel cell library as well as the special macros we have defined for this translation process.

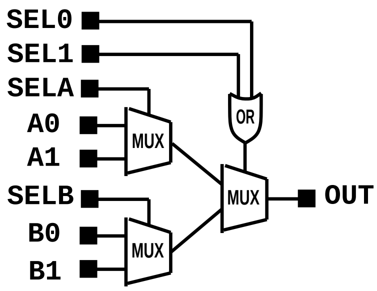

Figure 1: Actel Basic Logic Module

Chips from Actel contain some number of these logic modules, a collection of wiring segments, anti-fuses to connect those segments, and a set of programmable I/O pads. Examples of two common sizes of Actel parts are given in Table 1.

Because we use schematic capture and simulation tools from Viewlogic to design Actel chips [5], the output from our translator is a Workview netlist. If desired, this can be converted into a circuit schematic using the Viewgen program from Viewlogic which may be modified by the designer prior to programming the Actel part. The Workview netlist is also used by the ALS tools from Actel to map the translated circuit onto the Actel FPGA [6]. The ALS program performs pin assignment, placement and routing of the circuit onto the Actel chip, and generates the file with programming information for the anti-fuse connections. It also performs static timing analysis and provides delay back-annotation to the Viewsim simulation program.

The Workview schematic capture and simulation environment from Viewlogic combined with the Action Logic System (ALS) from Actel form a powerful design environment for quickly and inexpensively building small quantities of special purpose chips. The limitations of these FPGAs are in the size and speed of the circuits that can be mapped to the FPGA. Also, the cost is low if only a few chips are desired, but can become expensive if larger quantities are needed.

2.3 Resulting Environment

An overview of the translator environment is shown in Figure 2. Circuits are designed as custom circuits using either *ACME* or *Tiler* and simulated using *Simppl*. These circuits are converted, using our circuit translation rules, into Workview netlist format, and from there an Actel FPGA is generated. If simulation of the Actel chip is desired, it may be accomplished using the Viewsim simulator.

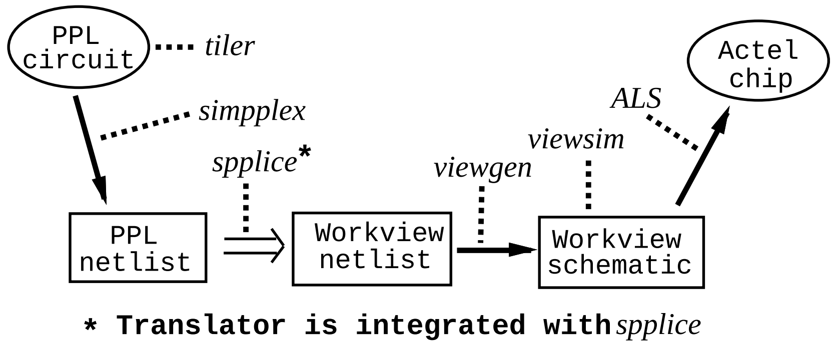

Figure 2: Overview of the translator environment

3 Example: A 4-bit Pipelined Multiplier

To demonstrate the translation process we have chosen an array multiplier built using a series of carry-save adders (CSA) with a carry-propagate adder, in this case a carry-lookahead design, in the final stage to resolve the carry-save result [7]. To improve the performance of the multiplier, and to test the translation procedure using various clocking schemes, we have inserted pipeline registers between each row of adders. We have also inserted an extra pipeline stage in the final carry-propagate adder to balance the pipeline stages in the multiplier. The resulting pipelined multiplier is controlled in two ways: as a conventional synchronous pipeline, and as an asynchronous micropipeline [8]. The circuits were designed to fit on a MOSIS Tiny Chip, so the circuit was limited to a four-bit multiplier which produces an eight-bit result. Figure 3 shows the block diagram of this multiplier.

Two MOSIS Tiny Chips were designed to serve as the source for the translation. These chips are:

Synchronous-MOSIS (SMO) A version of the pipelined multiplier controlled in a standard synchronous fashion. A global clock signal is used to control the flow of data through the pipeline registers.

Asynchronous MOSIS (AMO) A version of the pipelined multiplier where flow of data through the circuit is controlled asynchronously at each pipeline stage in the style of micropipelines [8]. A micropipeline uses no clocks and instead uses C-elements and two-phase transition signals to control the data latches locally in each pipeline stage. Data are inserted into the pipeline by making a transition on the input request signal, and the result of the multiplication is signaled by a transition on the output request line. Each stage of the micropipeline will latch new data when (i) the previous stage has new data to give and, (ii) the following stage is finished with the current data. In this way data moves through the pipeline asynchronously as each stage is finished with its own local processing.

Both of these chips have been fabricated, tested, and found to be fully functional.

4 The Translator

Because PPL circuits are built using a set of fine-grained library cells, it is a somewhat straightforward matter to define cell-by-cell translation rules to convert from PPL cells to Actel cells. This is exactly what we have done for the initial automatic translator. PPL constructs are given Actel equivalents and the circuit is converted using syntax-directed translation from one cell set to the other. In many cases, the PPL cells have a direct equivalent in the Actel cell library. For those cells that do not have a direct mapping, we have generated new Actel cells using the basic Actel macro shown in Figure 1.

The process of translating the PPL circuit into the Actel circuit begins with the PPL circuit extractor *simpplex*. This program extracts the circuit from the PPL layout by collecting PPL cells into standard logic gate descriptions. PPL circuits may use fine-grained cells to build distributed gate structures that can look like PLA rows and columns, although in a much less constrained form than in a PLA. These structures are collected into standard gates by *simpplex* and entered into a design database representing the circuit. Once in this form, translations of standard INVERTER, AND, NAND, OR, and NOR gates, for example, are straightforward.

A netlist translator called *spplice* is used to generate different output formats from this database description. Our translator uses syntax-directed rules to modify the behavior of the *spplice* netlist translator. This converts the circuit in the design database into an Actel equivalent. In particular, the Actel circuit is represented as a Viewlogic netlist.

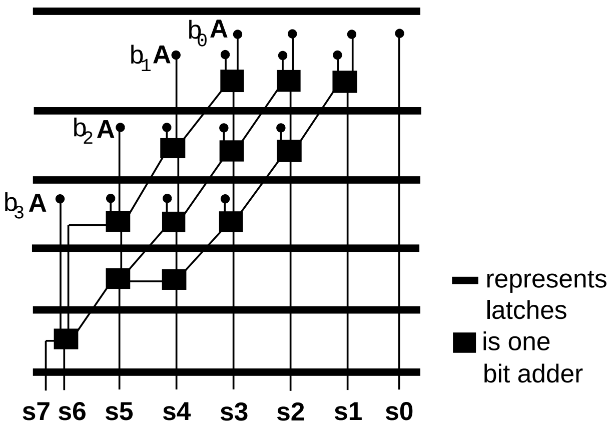

Figure 3: A four-bit pipelined array multiplier

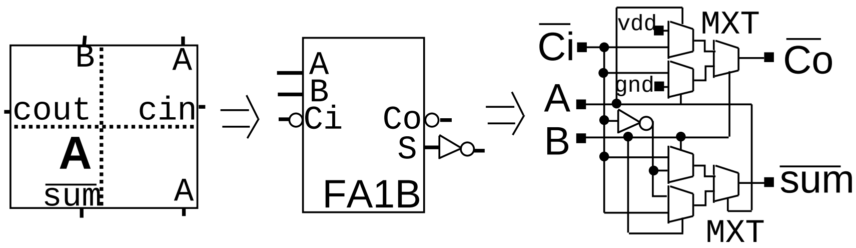

Figure 4: Translation of a full adder

4.1 Translation issues

In order to use simplified syntax-directed rules, the initial version of the translator places a small number of restrictions on the PPL designs. One restriction is required because the syntax-directed translation cannot adjust for variations in output drive in the PPL cells. The PPL cells have modifiers that allow variations in output drive from 1x (standard size) to 8x (eight times larger output drive) depending on the load. Because the Actel cells have uniform output drive equivalent to the lowest 1x drive of the PPL cells, for automatic translation the PPL cells are restricted to their 1x versions. This means that some cells might be unnecessarily duplicated in the PPL design to increase their output drive without using modifiers.

Another issue related to fan-in and fan-out is that some of the Actel cells present different fan-in characteristics then their PPL versions. This can cause the Actel circuit to violate fan-out restrictions in circuits that were correct in the PPL design. This has not been a problem in the examples we have tried, but a future version of the translator needs to take this into account.

4.2 Example Cell Translations

Full Adder Full adders are represented in PPL as single cells, the most efficient version of which use negative logic for the carry-in and carry out bits, and produce an inverted sum output. Because a cell of exactly this type does not exist in the Actel library, a new cell was designed for the translator. As shown in Figure 4, two different versions of the full adder cell are considered. The first is the Actel FA1B cell. This cell is a full adder that uses negative logic carry signals, but produces a positive sum. An extra inverter fixes this problem at the expense of increasing the delay of the sum signal by adding one extra level of logic. The second choice also uses three Actel basic macros, but has only two levels of logic between input and sum. Either adder cell may be used in the translation, although future versions of the translator may prefer one over the other depending on the structure of the surrounding circuit.

Clock Driver PPL clock driver cells generate two-phase non-overlapping clock signals. These clocks are used by a variety of PPL cells that require a multi-phase clock. In contrast, Actel cells that require a clock use only a single phase clock. Actel ACT-1 FPGAs provide a single dedicated low-skew clock line across the chip. In an Actel design, this would normally be used to distribute the system clock. However, for this syntax-directed translation, we cannot be sure that the PPL multi-phase clock is

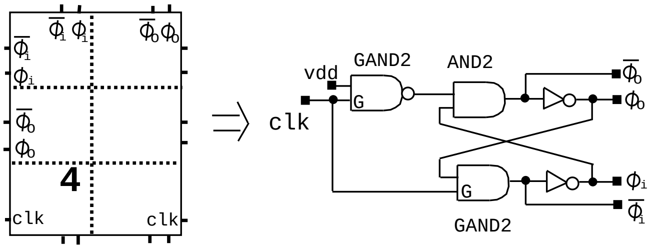

Figure 5: Translation of a clock cell

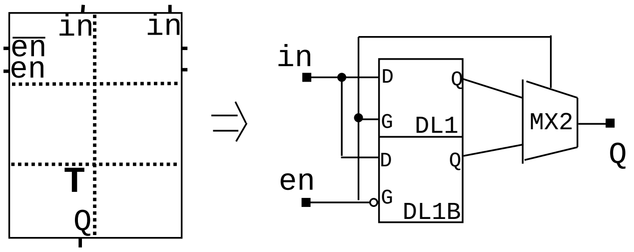

Figure 6: Asynchronous static latch translation

used only for flip flops. Instead, we must assume that the different phases might be used in different ways. So, the translation to an Actel chip implements a clock generator similar to the one in the PPL cell set as shown in Figure 5.

Latch cells Two different types of latch cells are used in the SMO and AMO PPL chips. A static inverting transparent latch is used in the synchronous SMO chip. This is translated directly to an Actel latch with similar characteristics. The main differences are in the clocking schemes. The PPL latch uses non-overlapping clocks from the clock generator cell while the Actel version uses a single phase clock. The asynchronous AMO chip uses a dual-edge-triggered latch that allows data to be captured on both edges of a clocking signal. This latch is implemented in Actel using three basic macros as shown in Figure 6.

C-elements C-elements are gates used frequently in asynchronous designs like the AMO chip [9]. A C-element will set its output low when both inputs are low, and set its output high when both inputs are high. If the inputs are in different states, the C-element holds its output at the previous value. This cell from the PPL cell set is implemented using three Actel macros as shown in Figure 7. Because the PPL cell offers both inverted and non-inverted outputs from the C-element, the Actel version also provides both output senses. However, if the inverted output is not used in the circuit, the optimization phase of the ALS place and route software will delete the dangling and extra inverter from the FPGA circuit.

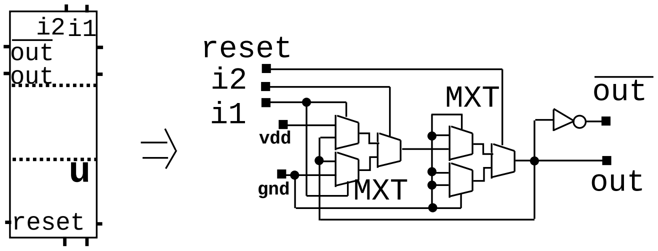

Figure 7: C-elment translation

5 High-Level Translation Issues

Although the current syntax-directed automatic scheme has the virtue of simplicity, the translations could be improved greatly by applying some higher level translation rules. These rules could improve the resulting FPGA circuit by eliminating many of the inefficiencies mentioned previously. The following are examples of the type of higher level rules that could make the translated FPGA circuit more efficient.

Delay chains In some PPL designs, especially asynchronous designs like the AMO chip, small chains of inverters may be used as delay lines. In the current translator, these inverters are translated one-for-one to Actel inverters. However, the different delay characteristics of the Actel chip mean that these direct translations will result in much more delay than actually needed. A high level translation could modify the delays to be appropriate to the situation.

Inverter elimination Using the PPL cell library generally results in using standard NAND and NOR gates for random logic in the circuit. The Actel library, in contrast, offers many alternate gates with inversions on selected inputs or the output of the gate. These gates could be used to translate equivalent logic functions to the Actel FPGA rather than translating the PPL gates exactly to the Actel chip. For example, inverters in the PPL chip can, in many cases, be subsumed into the following gate by using a version of that gate with an inverted input.

Flip-flops The dual-edge-triggered latches used by the asynchronous AMO chip are currently translated into a three-module circuit shown in Figure 6. With only a slight modification to the circuit, these asynchronous latches can be implemented in a form that uses only a single Actel macro. A synchronous flip-flop can be decomposed into AND/OR gates and two static latches, which can be individually translated.

Clocking mechanisms The clock generator circuit used in PPL can, as long as it is used only as the clock input to sequential elements, be translated to use the dedicated clock line on the Actel part. This does not change the functionality of the circuit, since the Actel latches use the single phase clock, but could make the FPGA circuit much more efficient.

Adder cells In PPL, the full adder cell is very highly optimized. So much so that it is used even when only a half adder is required. If this is the case, a high level rule could substitute the Actel half adder without changing the circuit and gain an extra module for each adder.

6 Example: Translated Actel Versions

Using the two PPL circuits SMO and AMO as a starting point, four Actel circuits have been generated. Two were generated automatically using the syntax-directed cell-by-cell replacement rules, and two translations were done by hand using more efficient high-level translations to determine the effect of a more sophisticated translation on the resulting FPGA circuits.

The Actel chips are:

Synchronous Rule Translation (SRT) An Actel version of the SMO synchronous chip translated automatically using the current rule base.

Asynchronous Rule Translation (ART) An Actel version of the AMO asynchronous chip translated automatically using the current rule base.

Synchronous Hand Translation (SHT) A version of the SMO synchronous chip translated by hand using high-level translation rules.

Asynchronous Hand Translation (AHT) A version of the AMO asynchronous chip translated by hand using high-level translation rules.

All the chips have been programmed, tested, and found to be fully functional. Furthermore, they faithfully mimic the behavior of the PPL chip from which they were translated.

7 Test Results

The two MOSIS Tiny Chips, SMO and AMO, were fabricated in the MOSIS 2-micron SCMOS technology. The Actel chips were programmed into A1010A or A1020A Actel parts depending on their size. All chips were functional and were tested for speed on a Textronics LV500 tester. The results are shown in Table 2. For the synchronous chips, the maximum clock is the highest rate at which the tester reported consistently correct operation. The minimum latency through the synchronous chips reflects the five pipe stages, and therefore five clock cycle delay for a single result to pass through the chip. The SRT Actel design using the simplified syntax-directed rules was a factor of 2.3 slower than the PPL chip, and the SHT hand translated version was only a factor of 1.3 slower.

For the asynchronous design, the slowest stage reflects the delay encountered in the logic between any two pipeline stages. The latency is reduced from the synchronous version because each stage in the asynchronous version takes only as long as it has to. The synchronous version requires that all

Table 2: Multiplier Performance Measurements

	sync. chips		
	SMO	SRT	SHT
Max Clock	24nS	56nS	32nS
Min Latency	120nS	280nS	160nS
	async. chips		
	AMO	ART	AHT
Slowest Stage	24nS	88nS	60nS
Latency Time	72nS	348nS	216nS

Table 3: Area Measurements for Multiplier Example

	SMO		AMO	
Chip Type	MOSIS tiny chip			
Max Cells	2618 unit cells			
Max Area	in 2.2mm X 2.2 mm			
Used Area	493 cells(logic) 949 cells(wiring)		700 cells(logic) 1189 cells(wiring)	
	SRT	SHT	ART	AHT
Actel Chip	1010A	1010A	1020A	1010A
Modules/Chip	295	295	547	295
Modules Used	250	187	385	200
Area Ratio	SRT/SHT = 1.34 , ART/AHT = 1.93			

five stages in the pipeline take the same time. In fact, the carry-save adder stages take less time than the carry-lookahead stages at the end. The asynchronous multiplier can take advantage of this to show reduced latency through the multiplier. The Actel versions of the asynchronous multiplier show slowdown of 3.6 and 2.5 for the slowest stage, and 4.8 to 3.0 for the latency between the rule and hand translated versions. This extra slowdown for the latency is caused in large part by inefficient translation of request/acknowledge control circuitry.

This speed measurement result shows one interesting performance comparison between synchronous circuits and asynchronous ones based on equivalency : Synchronous versions have better thgoughput, and asynchronous versions better latency.

The area measurements for the example chips are shown in Table 3. An interesting measure is the ratio of the rule-translated Actel chips to the hand-translated Actel chips. This shows that in addition to improvements in performance, the high-level translation rules improve the size of the resulting Actel prototype chips by a significant factor as well. Although it is difficult to compare chip capacities that use different technologies, this also indicates that a PPL tiny chip has approximately the same functionality as an Actel 1010A or 1020A chip. This will, of course, depend heavily on the structure of the circuit being implemented.

8 Conclusions

We have described a procedure to allow custom VLSI designs to be prototyped quickly and inexpensively using FPGAs. Specifically, we have shown a technique for translating custom PPL chips into Actel FPGAs. By designing the custom chip first as the source of the translation, and mapping that custom chip automatically to an FPGA, we have avoided the problem of technology mapping that is often encountered when trying to design the prototype first and map to a custom technology only after the prototype is found satisfactory.

Using PPL synchronous and asynchronous pipeline multiplier chips as examples, we have demonstrated both an automatic syntax-directed translation and a more sophisticated translation that uses some higher level translation rules. Six chips were produced. Two MOSIS chips were fabricated, and four Actel FPGAs were programmed. These chips were tested and all were found to be fully functional. Furthermore, the FPGA circuits were found to be faithful copies of the original PPL circuits. Testing results for these examples show that the FPGA prototype performs at between 43% and 75% of the speed of the synchronous CMOS custom chip and from 21% to 40% of the asynchronous custom chip's speed.

References

[1] Kent F. Smith and Jun Gu. A Structured Approach for VLSI Circuit Design, *IEEE Computer* , Vol 22, No. 11, November, 1989

[2] Tony M. Carter, Kent F. Smith, Steven R. Jacobs, and Richard M. Neff. Cell Matrix Methodologies for Integrated Circuit Design, *Integration, The VLSI Journal* , July, 1989

[3] Actel Corporation. *ACT Family Field Programmable Gate Array Databook*, April 1992

[4] Steven R. Jacobs. *PPL Database Reference Manual*, January, 1993

[5] Viewlogic Systems Inc. *Schematic Design User's Guide, Version C, For use with Workview 4.1, Series II* , May 1991

[6] Actel Corporation. *Action Logic System(ALS) Release 2.1 User's Guide*, August 1991

[7] John L. Hennessy and David Patterson. *Computer Architecture, A quantitative approach*, Morgan Kaufmann Publishers, Inc., 1990

[8] Ivan E. Sutherland. Micropipelines, *Communications of the ACM*, June 1989

[9] Erik Brunvand. A cell set for self-timed design using Actel FPGAs, *Tech. report UUCS-91-013*, The University of Utah, 1991