AFRL-IF-WP-TR-2001-1514

OPTIMIZING VHDL INTERMEDIATE FORMS

Dr. Keith Cooper

Rice University
6100 South Main
Houston, TX 77005

MARCH 2001

FINAL REPORT FOR PERIOD 16 SEPTEMBER 1997 – 17 SEPTEMBER 2000

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

NOTICE

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

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

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

KERRY L. HILL
Project Engineer
Hardware Team
Embedded Info Sys Engineering Branch

ALFRED J. SCARPELLI
Team Leader
Hardware Team
Embedded Info Sys Engineering Branch

JAMES S. WILLIAMSON
Chief
Embedded Info Sys Engineering Branch

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

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2001	Final Report, 09/16/1997 – 09/17/2000

4. TITLE AND SUBTITLE
Optimizing VHDL Intermediate Forms

5. FUNDING NUMBERS
C: F33615-97-C-1125
PE: 69199F
PR: ARPA
TA: AS
WU: 11

6. AUTHOR(S)
Dr. Keith Cooper

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Rice University
6100 South Main
Houston, TX 77005

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7334
POC: Kerry L. Hill, AFRL/IFTA, 937-255-6548 x3604

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-WP-TR-2001-1514

11. SUPPLEMENTARY NOTES

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

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
This program was based on leveraging software compiler optimization expertise to further improve upon hardware compiler optimization techniques for Field Programmable Gate Arrays (FPGAs).

14. SUBJECT TERMS

15. NUMBER OF PAGES
16

16. PRICE CODE

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

NSN 7540-01-280-5500

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

Optimizing VHDL Intermediate Forms
– *Final Report* –

Keith D. Cooper, Project Director
John Bennett & Linda Torczon, Principal Investigators

Rice University
ARPA Order F223
USAFRL Contract F33615-97-C-1125

`http://www.cs.rice.edu/~keith/VHDL`

1. Overview

This project began with a simple motivation and a simple goal. We observed, through our classroom experience, the poor quality of the circuits synthesized from student-written VHDL programs (See the appendix). Based on our knowledge of compilation techniques for software, we felt that this poor circuit quality most likely resulted from flaws in the translation process. Our proposal laid out a project to build a prototype optimizer for VHDL-derived circuits. It relied, in a critical way, on the fact that commercial design tools allow the user to export a circuit in some standardized intermediate representation (IR). Our plan was to (1) select one of the available intermediate representations, (2) obtain tools to import and manage that representation, (3) build some transformations that manipulated the representation, and (4) export it back into the tools. As part of the process, we would gather a suite of VHDL codes that could help assess the quality of translation in VHDL tools.

As the project progressed, two forces led us away from this plan.

- The infrastructure task was larger than we anticipated. Publicly available tools did not exist for the industry standard IR, EDIF; thus, we had to build them. To further complicate matters, EDIF is purely syntactic; this makes library understanding a critical issue—one that we did not intend to have on our plate.
- Dr. Munoz, of DARPA, asked us to investigate other issues. These included the impact of IR transformations on place and route time, and the possibility of improving circuits that had already been mapped to an FPGA.

As the development work progressed, it became apparent to us that building a truly useful optimizer would require access to the kinds of tools available only to the actual developers of the design tools—source code for the tools and the cell libraries that they use. In the end, we demonstrated that our ideas would work, but that the project would require a much larger development effort than either our budget or our organization could support.

In the end, this project demonstrated that our basic insights were sound. Viewing the intermediate form of the design as a program and operating from that analogy allowed us to simplify and improve circuits. In particular, it allowed us to simplify the example that motivated the original proposal (see the Appendix). Along the way, we showed that

reordering the IR file could improve place and route times[1], and that we could squeeze some additional logic out of already mapped circuits. At the same time, the sheer amount of infrastructure required to perform this work overwhelmed our effort. We were unable to produce generally applicable tools, because the programming burden was much larger than we anticipated or than our budget would support. To make matters worse, optimization on the IR requires a detailed knowledge of the contents and characteristics of the vendor's cell libraries. This is simply unavailable to academic researchers.

2. Results

The fundamental idea behind this project was that we can apply the insights derived in classic scalar compiler research to the kinds of inefficiencies that arise in circuit designs generated from VHDL programs. Our goal was to establish an analogy between these designs, in IR form, and programs. Our approach was to view the circuit as a graph that specifies a computation; to analyze the flow of values through the graph; and to transform the graph in ways that improve it.

We produced results in three different arenas: the classic optimizations that we originally proposed to study, the potential for preprocessing designs to reduce their place and route times, and the potential for improving circuits after they had been mapped to an FPGA.

2.1 Classic Optimization: We built a prototype system that would parse an EDIF program, build a graph that instantiated the design embodied in the program, apply an optimization to that graph, and report the results. We designed and implemented a tool to take the graph back into a valid EDIF program, but were unable to fully debug that part of the system (see §3.3).

The first optimization that we implemented was an adaptation of Balke's classic value numbering algorithm [1]. It discovers redundant values, folds constants, and provides a framework for applying algebraic simplifications. It required numerous changes to work as desired on VHDL-derived circuits. The original algorithm tracks values. The new algorithm understands that each value has a complement, that those values are related, and that it must track them as a pair. The new algorithm understands an inverter as an operation on value numbers, rather than as a graph element that produces a new value. It simplifies multiple negations. In the VHDL-derived designs, operations have variable numbers of inputs (AND cells, for example, come in many flavors), and multiple outputs; neither of which arise in the IRs used to compile software. Finally, the algorithm must deal with nodes that represent buffers, pads, and other cells that are important to the design but do not have an impact on values.[2]

With these extensions, the algorithm was able to simplify VHDL-derived designs. In our motivating example, it could recognize that the two latches were redundant and replace

[1] Many heuristic methods for approximating answers to NP-complete problems are order sensitive. In our experiments, the consistent improvement came from simply reversing the order in which the IR was generated by the tools.

[2] To further complicate matters, a PAD typically has one connection. The algorithm must recognize that some cell connectors are ambiguous, being both in puts and outputs. EDIF libraries complicate this kind of analysis by not using the EDIF attributes INPUT, OUTPUT, and INOUT consistently for cells like these.

the pair of them with a single latch. The transformation found redundancies in many designs. As in the motivating example, they appear to arise as artifacts of the translation process; as the compiler fills in the details to implement some particular VHDL construct, it misses opportunities for reuse. Just as in software compilation, these inefficiencies can be remedied with reasonably simple optimizations—analysis followed by transformation.

As a second optimization, we implemented a simple peephole optimizer. It examined a single node and its successors, looking for simplifications through simple pattern matching. This routine is small, efficient, and easy to implement. The patterns must be hand-coded.[3] The peephole pass is useful for finding and combining trees of identical operations. In one example, taken from VHDL produced outside Rice, the peephole optimizer turned an 8-input tree of binary AND operations into an eight-way AND cell.

The demonstration, in principle and in simple prototype, that classic compiler optimization techniques can improve the designs produced by commercial VHDL compilers was a major result of this project. When we began this project, this base proposition was greeted with deep skepticism. Our experiments demonstrated that value numbering, negation propagation, and peephole optimization.

2.2 Place-and-Route Work: In the fall of 1998, Dr. Munoz asked us to look at the impact that our ideas could have on the time required for place-and-route with VHDL-derived designs being mapped to FPGAs. By this point in the contract, we were wary of taking on additional infrastructure development, so we took a two-track approach to answering the question. On one path, we performed an experiment to determine how robust the commercial tools (that we had) were with regard to place-and-route time. On the other, we considered the ways that optimization might influence place-and-route time.

Our experiment was simple. We know, from experience with heuristic approximations to the solution of NP-complete problems, that the order in which a problem is presented often affects the behavior of tools that try to approximate their solution. Thus, we had reason to suspect that reordering the IR representation of a circuit might change the time required to place it and route it.

To test this hypothesis, we built a simple reordering tool for the EDIF representation of the program—the EDIF abstract syntax tree produced by the parser. The tool recreates the circuit in three new orders. It orders the nets by increasing size—that is, number of connections. It orders the nodes by decreasing size. Finally, it creates a version in the opposite order of the code produced by the vendor's tools. Using the Viewlogics tools, we compared the place and route times for twenty distinct designs. They ranged from small designs (5 to 50 CLBs) where placing and routing took a couple of seconds, through moderate-sized designs (1,400 to 1,500 CLBs) that took between 200 and 600 seconds to process.

[3] The best peephole optimization systems, for software compilation, work from specifications that describe the IR and the target machine's instruction set. Some perform a large amount of preprocessing to find patterns. Others discover patterns at compile time. Since our goal was to demonstrate the concept, we built a simple system with hand-coded patterns.

On the small designs, the time to place and route was insensitive to the order of presentation. This was as expected, since these designs occupy a small portion of the FPGA. The six largest designs, which include the Scalability FFT codes from the ACS Benchmark suite, showed order-dependent variation in place-and-route time. We saw improvements of twenty to thirty-five percent, and one case of degradation by twenty percent.

At the suggestion of reviewers (2/99 program review in UC Irvine), we repeated the tests to look for random variation. This kind of variation might be expected if the tools used methods such as simulated annealing to perform place-and-route. The suggestion, from reviewers, was that our improvements might simply be expected variation due to randomization in the place-and-route tool. We repeated each test twenty times; this produced no significant variation on either the small designs or the large designs.

In every case, the observed variation from orders based on net size was minimal. All the major improvements resulted from the reversed order—that is, listing the nets in the opposite of the order generated by the vendor's tool. The vendor's tool generates the file in a hierarchical fashion, so it lists the nets inside a cell before the nets that connect that cell externally. Thus, the reverse order presents the nets in an order from outside to inside. This corresponds, roughly, to most constrained net first, or net that connects largest cells first. This forces the place-and-route tool to consider larger cells and larger nets first, and smaller, less constrained cases last. This is a well-known heuristic that works on other problems; it corresponds to the order used for coloring a graph in a Chaitin-Briggs register allocator [2].

To apply these insights in a systematic and thorough way, we devised a strategy that would use value numbering to shrink the design's graph to a minimal number of cells. Next, it would analyze the graph's connectivity to discover cells that should be replicated to simplify routing. The tool should replicate cells to simplify routing, until it reaches a threshold design size, such as 90% of the FPGA capacity. Finally, it would emit the EDIF for the design in a carefully chosen order, specifying nets in an outside to inside order. Unfortunately, the infrastructure difficulties prevented us from finishing the tools to conduct this experiment. The examples that we could get through our tools were too small to be conclusive.

2.3 Optimizing Mapped Circuits: At the April 1999 PI Meeting, in Houston, we took on another challenge. During the poster session, several people suggested that we look at the problem of improving circuits after they had been mapped onto an FPGA. Dr. Munoz asked us to look at the issue. This led to another (unanticipated) investigation, which formed the core of a master's thesis for Vasileanos Balabanos (Rice MS, ECE, 2000).

To handle mapped designs, we first developed an algorithm for discovering subsets of the design that form sequential circuits. Typically, this is an acyclic region within the design that is set off from the rest of the design by one or more latches on every data path. Once the sequential subregions have been identified, we can derive a set of equations that describe their behavior. These equations are fed to an external simplifier that uses symbolic methods to transform the equations so that they have fewer terms. The simplified equations can then be re-synthesized into an improved design.

As part of his thesis work, Balabanos performed this transformation on several mapped circuits. In some of the circuits, it found no opportunity for improvement. In others, the simplifier was able to remove one or more CLB from the design.

This work relied on an external simplifier from UC Berkeley. Our contribution was the algorithm to discover and isolate sequential subgraphs in the design and to extract the equations. The experiments were done with a combination of automatic and manual techniques. Time constraints imposed by Balabanos' graduation, plus the reliance on external software that we can use but cannot redistribute made it impractical to develop a fully automated tool for distribution.

3. Lessons

In the course of this contract, we learned many lessons. Several of them are significant and should be reported to help others learn from our experience.

3.1 Validity of the base concepts: All of our work and our experience in this contract showed that our base concept was valid. The translation phases of commercial design tools have many of the characteristic problems that arise in compiling Algol-like programming languages. Viewing the designs as graphs that represent abstract computations, and subjecting those graphs to transformations that find redundancy, that discover and propagate known values, reason about algebraic notions of identity, and that combine operations can lead to simpler, smaller designs.

Our experience suggests that optimizations such as value numbering, constant propagation, and peephole optimization complement the boolean simplification techniques used in most commercial design tools. Applying a combination of techniques that draw from different perspectives should produce better circuits.

Design optimization should have two distinct goals. First, optimization should improve the overall quality of the results produced by design tools. Second, optimization should broaden the set of inputs that produce high-quality results. Optimization should make it possible for design tools to create high-quality results from behavior specifications or from naively-written specifications. A well-done design optimizer should reduce the extent to which the quality of the output depends on the form of the input.

Using techniques like those that we explored should help in this regard. Our value numbering tool was able to overcome limitations in the initial translation of VHDL into netlist form. On the motivating example, it recognized, through careful reasoning and a little algebra, that the design needlessly duplicated a computation. In practical terms, it recognized that a flip-flop can be implemented with a single latch, rather than one latch per output. Applied to larger circuits, like those in the ACS benchmark suite, the same tool found additional redundancies. The peephole optimizer found other artifacts of translation, such as trees of binary operations that it replaced with a single, multi-input operation.

Our work suggested that the right model for design optimizations is to use optimistic global techniques rather than local techniques [3]. The designs that we encountered had graph structures that were complex enough to confound local analysis. Similarly, optimism will aid in the analysis and transformation of cyclic graphs.

3.2 Choice of Intermediate Representation: In the first six months, we studied the available options for an intermediate representation (IR) and selected EDIF. In retrospect, this was a critical decision. Despite any negative statements that this report makes about EDIF, our decision to use EDIF rather than other available IRs was probably the correct one.

The critical factors in this decision were:

- We needed an IR that was supported by commercial tools. Without the ability to generate IR from commercial systems, our experiments would lack credibility.
- We needed an IR that had a detailed, low-level view of the design. The techniques that we explored produce their benefit by accumulating many small wins, not by finding one "home run" improvement. This style of cumulative improvement works better with more detail than with less detail.

  (However, detailed, low-level IRs usually require more complex programs to manipulate them. This is a natural consequence of having to deal with that many more cases at every turn. Thus, using a low-level IR like EDIF made the programming task larger. We were, frankly, surprised by the size of EDIF. Its grammar dwarfs programming languages like Fortran, C, or Java.)

To make the decision, we studied the available documentation for the available systems. We obtained the available tools for several systems, and did small-scale explorations of the software to determine whether or not it was suitable for our use. (A typical exploratory project was to get the system in-house, feed some small VHDL designs through it, and isolate the tools required to read and write the IR. With the OCEAN system, from TU-Delft, for example, this exploration took three weeks.) We discovered that thepublicly-available tools that used EDIF supported subsets, and that those subsets excluded many features that are used in practice.

After evaluating the available choices and discussing the matter with Dr. Munoz, we decided to proceed with EDIF as the IR. Unfortunately, this meant developing our own tool base, since we could not locate a publicly-available parser for the full EDIF 2.0.0 language. (Our parser is available on the project web site.) EDIF has the right combination of commercial tie-in and level of abstraction. As we realized later, it also has some major problems for the kind of work that we tried to do.

(The primary problem with EDIF is that the language is purely syntactic. That is, an EDIF program expresses the interconnections between a series of cells. An EDIF program has little or no inherent meaning. The knowledge needed for any non-trivial level of optimization is buried in the libraries. To understand the effects of a cell on the values that flow through it requires an understanding of the differences between cells. Unfortunately, in EDIF, a two-input AND cell and a two-input OR cell have the same description, except for the cell names. Thus, to optimize the design in EDIF requires a detailed understanding of the library.

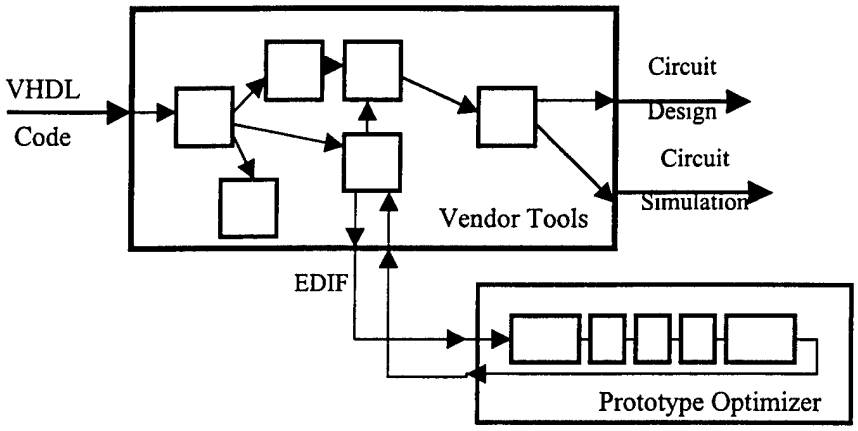
Since the libraries are vendor-specific and device-specific, any attempt to manipulate the EDIF form of a design turns into a library understanding problem. In practice, this gave us a huge unanticipated subproblem: deriving the needed knowledge from the limited available information on these vendor libraries. In the end, our tools still needed manual

intervention to add detail such as whether or not a given operation is commutative—to answer questions like "is x AND y equivalent to y AND x?")

Despite these issues, EDIF was probably the correct choice. Still, this work would have made more progress if it had faced a smaller infrastructure problem.

3.3 Reflections on the project's design: From the beginning of the project, several reviewers were skeptical about our original experimental design. The criticism included suggestions that we should attack the problem at a higher level of abstraction—perhaps using an abstract-syntax tree for VHDL—and suggestions that the kinds of improvement we sought would not be found in the output of commercial tools. The success of our limited experiments suggests that low-level optimization of VHDL-derived designs, following our language-centric paradigm, can work, and that the opportunities are, indeed, created by commercial tools. However, the limited nature of our experiments testifies to a deeper flaw in the project's design—one that seriously impeded our progress.

Our design, shown below, relied on our ability to take designs, expressed in EDIF, from the commercial tools, modify it, and re-insert it into the work flow of the tools. In theory, this is straightforward. In practice, it proved difficult. If the prototype optimizer works



perfectly, then the design works. However, the setup provides little or no support for debugging the optimizer. If the optimizer produces a design that the vendor tools will not accept, there are no tools to help the compiler writer understand either the nature of the problem or its specific details. Furthermore, at that point, the optimizer has obliterated any correspondence between input and output by renaming, rewriting, and reordering the design.

The nature of EDIF accentuated this problem. The lack of deep semantics for EDIF and the heavy reliance on vendor-supplied, device-specific libraries complicated the debugging process for tools in the optimizer. The implementations of specific cell libraries can exacerbate these problems. Several libraries were cavalier about designating the types for external connections to buffers and pads. The inputs were declared variously as input ports, as in/out ports, and as unknown ports. A deeper semantics for EDIF would require consistent treatment of these ports.)

We encountered this kind of problem many times in the course of the project. In the end, it convinced us that to successfully pursue this kind of work, we would need access to the source code for the vendor's tool set and to detailed information about their device-specific cell libraries. Without such insider's knowledge, the debugging process rapidly degenerated into a frustrating hunt for undocumented minutia. The largest factor that impeded the progress of our software development efforts was the difficulty of debugging tools in the optimizer. In retrospect, we should have anticipated this. In practice, it was the Achilles' heel of the project.

4. Deliverables

This project produced a number of concrete results. Some are available in software form, while others appear in technical reports or papers intended for publication.

1. The most useful software artifact of the project is a parser for the full EDIF 2.0.0 language. The parser is written in C, using standard LR(1) parser generator tools (yacc). It has been available on our web site for roughly two years. Other groups, both in the DARPA ACS community and beyond, have used the parser.

   (We are currently updating the documentation for the parser to make it more useful. New documentation will be posted on the web site as we finish it.)

2. For the prototype optimizer, we produced a value numbering tool that is fairly complete. W e are in the process of cleaning up that code. It will appear on the web site in the first quarter of 2001.

   At the same time, we are writing a technical report on the adaptation of value numbering to EDIF-based circuit designs. The technical report will appear on the web site when it is finished.

3. We performed a study showing that the place-and-route times of commercial tools are quite sensitive to the order in which a design is presented. We delivered the results of this study in talks and poster sessions, with copies of the presentations transmitted to contract monitors at both USAFRL and DARPA ITO.

4. We developed an algorithm for optimizing circuits that have already been mapped. This work is described in Balabanos' master thesis.

5. As the contract progressed, we relied heavily on the ACS Benchmarks developed as part of the DARPA ACS Program, supplemented by some examples that Dr. Munoz supplied and a small collection of hand-written examples. We do not have permission to redistribute those VHDL codes.

References and Notes

[1] The original value-numbering algorithm is credited to Balke, in the late 1960s. He did not publish a description of the algorithm, but contemporaries describe it. For a concise explanation, see "Value Numbering," by Briggs, Cooper, and Simpson, in *Software–Practice and Experience*, 27(6), June, 1997.

[2] "Improvements to Graph Coloring Register Allocation", Briggs, Cooper, and Torczon, in ACM Transactions on Programming Languages and Systems, 16(3), May, 994.

[3] Optimism refers to a property of static analysis algorithms. Wegman and Zadeck introduced the term in their 1991 paper on constant propagation (ACM TOPLAS, 13(2), April, 1991). A more constructive definition appears in Click's Ph.D. thesis (Department of Computer Science, Rice University, February, 1995). For insight into the role of optimism in value numbering, see Simpson's Ph.D. thesis (Dept. of Computer Science, Rice University, April, 1996)

Appendix: The Motivating Example

Our original example was the following simple program, written in behavioral VHDL. It describes an RS flip flop.

```
ENTITY rs IS
        port (s,r: IN STD_LOGIC := '0';
            q,qn: OUT STD_LOGIC);
END rs;
--
ARCHITECTURE behavior OF rs IS
BEGIN
        PROCESS(s,r)
        BEGIN -- the ff will not change if r and s are both '0'
                IF s = '0' AND r = '1' THEN
                        q <= '0';
                        qn <= '1' ;
                ELSIF s = '1' AND r = '0' THEN
                        q <= '1' ;
                        qn <= '0' ;
                ELSIF s = '1' AND r = '1' THEN
                        q <= '1' ;
                        qn <= '0' ;
                END IF;
        END PROCESS;
END behavior;
```

The Viewlogics toolset produced the following circuit; notice that it uses two latches to implement a simple flip-flop. This circuit uses at least twice the logic that is needed. Our value numbering pass recognized that the Q output of LATCHED_VARIABLE_QN_2 had the same value as the Q' output of LATCHED_VARIABLE_QN_1, and eliminated the second latch. This reduced the size of the circuit significantly. However, with a better understanding of the library, it could have recognized that the cell that implements the latch also has RESET and SET inputs, so it could have (and should have) replaced the logic, minus input and output buffers and pads, with a single cell.