LOGIC SYNTHESIS FOR ASIC: A GUIDED ALGORITHMIC APPROACH

Michael Q. Le, Vojin G. Oklobdzija
Department of Electrical and Computer Engineering
University of California
Davis, CA 95616
vojin@ece.ucdavis.edu
(916) 752-5634

ABSTRACT

In this experiment, several circuits are designed using logic synthesis based on a behavioral model and a description based on a structured algorithmic approach. The results produced by logic synthesis based on the two different VHDL description styles are compared. The circuits are generated using Synopsys [2] tools and LSI Logic 300K ASIC technology as the target library [1]. The implementation based on the structured algorithmic description showed considerable improvement over the results produced by the behavioral model. With this experiment, we identify and point out the weaknesses associated with logic synthesis. The purpose of this experiment is to show that logic synthesis tools should not be used "blindly" and that imposing hierarchy and modularity in the design is very important in terms of regular layout and efficiency. Given that identification of hierarchy and modular design require an intelligent analysis, we feel that this is the major weakness of the logic synthesis approach in design. We are able to point out a direction for improvement and creation of intelligent logic synthesis tools.

1. INTRODUCTION

As a basis for comparison, the Leading Zero Detector circuit is selected. It is a circuit common to any *normalization* operation. Normalization consists of an appropriate left shift until the first non-zero digit is in the left-most position. The amount of shift is determined by counting the number of zero digits from the left-most position until the first non-zero digit is reached. A special circuit implemented (in hardware) to detect the number of leading zeros is referred to as a *Leading Zero Detector* (LZD) [3]. A characteristic of the LZD circuit is that its functional description is very concise. This fact makes it is very easy to describe the circuit using any of the VHDL Hardware Description Languages. On the other hand, design of such a circuit (especially for a large number of bits) is very cumbersome and difficult if no structure or algorithm is imposed in the design. Applying a straightforward combinatorial approach in designing the LZD circuit is a rather complicated process. The reason behind this problem is that each bit of the result is dependent on all of the input bits. In the case of a 32-bit word, the LZD would have 5 outputs, each dependent on 32 inputs. It is obvious that such large fan-in dependencies are a problem and that the resulting circuit is likely to be complicated and slow. To design such a circuit using computer aided Boolean minimization techniques or the Karnaugh map method is cumbersome and slow, and the resulting design does not exhibit any structure. On the other hand, this circuit is a very good candidate for *Logic Synthesis*. The functionality of the circuit can be described in VHDL, and a computer can do the rest of the work. However, we can use some insight and cleverness in designing the circuit. This particular circuit is quite suitable for exploring possibilities of hierarchical and structural design. Imposing hierarchy and structure would bring forth substantial improvement of the circuit regularity and speed, as compared to straight forward minimization. The resulting circuit is characterized by good performance and low and regular fan-in, which leads to better wireability and layout. Logic synthesis tools have not yet reached a level of sophistication to where they can deal with hierarchical structures or create and impose hierarchy in the design. Rather, their approach is to expand the logic in one level and optimize it via elaborate and laborious logic minimization, often using hours of CPU time. Therefore, the design presented in this paper not only results in an efficient and fast LZD but also provides an efficiency measure of the Logic Synthesis tools and their limitations [3].

2. DESIGN USING AN ALGORITHMIC APPROACH

We used the inherent hierarchy associated with the leading zero detection process in order to create a hierarchical and modular design. In the design process, we skipped the 2-bit LZD blocks and build the LZD circuit starting with 4-bit blocks, which eliminates one extra level. Each 4-bit block has one "V" output bit and two "P" output bits. The "V" bit is '0' when the four input bits are '0000' and is '1' otherwise. The "P" bits output the number of zeros encountered in that portion of the input vector. We can now take two 4-bit blocks to form an 8-bit LZD circuit using an 8-bit block that implements the algorithm shown in Figure 2.0. This concept can be expanded up to higher levels to form LZD circuits of any 2^N size [3]. The circuits produced from the algorithmic description are only one logic level deep, resulting in very fast N-bit blocks. The description for the 16-bit block and the resulting circuit produced by Synopsys tools are shown in Figure 2.1 and Figure 2.2, respectively.

3. LOGIC SYNTHESIS EXPERIMENT

We implemented two sets of LZD prototypes of sizes 16, 32, and 64 bits that were built from the two different VHDL description styles. The first set of circuits is synthesized (using the Synopsys Design Compiler) from its straight-forward *behavioral* description. The second set is synthesized using a guided and structured description based on the previously described algorithm. The results of those two approaches to logic synthesis are compared in terms of their mapping into a target technology. The target technology in our case is LSI Logic LCA300K [1]. Though the results might vary from case to case and with the target technology used, we feel that the general principles and results will remain the same.

3.1 SYNTHESIS USING BEHAVIORAL DESCRIPTION

The LZD circuit is described by a behavioral VHDL model. This description is then synthesized using the Synopsys Design Compiler and LSI Logic 300k technology as the target library. The *if-then-else* construct is used to describe the circuit, which results in a concise functional model of the circuit behavior. The description of the 32-bit LZD is given in Figure 3.1.

3.2 SYNTHESIS USING GUIDED STRUCTURAL DESCRIPTION

In this case, we guide our VHDL description with an algorithm for efficient generation of the LZD circuit. We impose hierarchy in the design by building several components which implement the algorithm and then map them together to form the LZD. This is done by describing the functionality of the individual blocks in VHDL, synthesizing them using Synopsys tools, and then mapping them together using the structural VHDL description style. We start the hierarchical structure by using 4-bit LZD blocks. The outputs of the 4-bit blocks are then connected to the 8-bit blocks, and so on, which implement the algorithm. For the 32-bit case, the blocks are mapped together to form the hierarchical structure shown in Figure 3.2.

4. PERFORMANCE

For the 32-bit LZD, the "behavioral description" resulted in an area of 564 normalized cell units (where 2-input NAND area is treated as one cell unit) and a nominal delay of 4.35 ns while the "guided structural description" yielded 426 normalized cells and a delay of 2.73 ns. The improvement in using the guided structural description is exhibited by 24% area reduction and 37% speed improvement for the 32-bit case. The other cases, 16 and 64 bit, yielded improvements using the guided structural description as well. The improvements in performance, measured in terms of nominal delay and total area, increase as the LZD bit size is increased. Results for the two description styles are shown in Table 4.1 and Table 4.2. It is important to note, however, that in certain cases, the logic synthesis tools produce very good circuits from the behavioral descriptions. In the case of implementing small finite state machines, such as simple controllers, the logic synthesis tools can produce smaller and faster circuits than the ones realized by trying to apply structure. The reason for their efficiency, in these particular cases, is that the heuristics used by the logic synthesis tools can easily deal with minimizing the number of states and covers needed in relatively small designs. But, in more complex cases where there is a fairly large number of inputs, like the LZD example, the heuristics used are simply not enough to solve the problems efficiently.

5. CONCLUSION

In this paper, we have compared the performance of circuits produced by logic synthesis tools for two different design approaches. The first one involves a straight forward behavioral model that is concisely translated into VHDL and then synthesized. The second one is based on a critical analysis of the problem and an identification of a proper algorithm and structure. This approach results in a description that guides the synthesis process and imposes hierarchy on the design. For all cases, the implementations based on the guided structural descriptions resulted in considerable advantages over the results produced by the purely behavioral models. The improvements over the behavioral models increased as the size of the LZD circuit was increased. With this experiment, we identify and point out the weaknesses of logic synthesis. We have shown that logic synthesis tools should be used judiciously and that imposing hierarchy and modularity in a design is very important in terms of area and speed. The lessons learned apply not only to this particular design, but generally indicate that careful analysis of a problem and clever management of the hierarchy pays big dividends in terms of the performance of critical circuits. Although very useful, logic synthesis tools are still not capable of managing hierarchies or making intelligent choices when it comes to design. Therefore, they should be used and treated accordingly.

REFERENCES

[1] LCA300K Compacted Gate Array Products Databook, LSI Logic Corporation, October 1993.
[2] Synopsys Inc., Synopsys Design Analyzer Reference Manual, Version3.0b., June 1993.
[3] V.G. Oklobdzija, *"An Algorithmic and Novel Design of a Leading Zero Detector Circuit: Comparison with Logic Synthesis"*, IEEE Transactions on VLSI Systems, Vol.2. No.1., March 1994.

Algorithm for generating LZ count:

(1) Form the pair of bits B_i, B_{i+1} for i=0 to N-2

(2) Determine P and V bits for each pair

(3) For the next level, determine the Pg and Vg bits as a function of two pairs of inputs P and V in this level in the following way:

$V_g = V_l + V_r$ where "+" is the OR operation of the left and right inputs

if *$V_l = 1$* ***then*** *$P_g = 0, P_l$ where "," designates concatenation*

else *$P_g = 1, P_r$*

Repeat step (3) log(N) -2 times

Figure 2.0. Algorithm for generating LZD.

```
entity comp16 is
    port ( PL, PR  :  in BIT_VECTOR (2 downto 0);
           VL, VR  :  in BIT;
           PG      :  out BIT_VECTOR (3 downto 0);
           VG      :  out BIT );
end comp16;

architecture func16 of comp16 is
begin
    process ( PL, VL, PR, VR)
    begin
        VG <= VL or VR;
        if ( VL = '1' ) then
              PG(3) <= '0';
              PG(2 downto 0) <= PL(2 downto 0);
        else
              PG(3) <= '1';
              PG(2 downto 0) <= PR(2 downto 0);
        end if;
    end process;
end func16;
```

Figure 2.1. VHDL description of the 16-bit block.

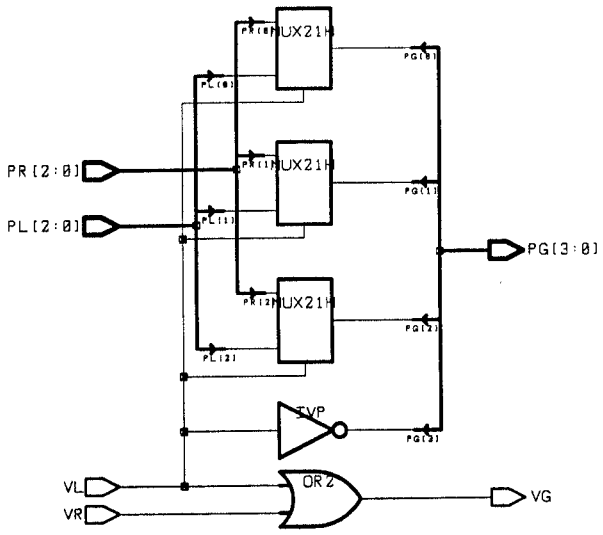

Figure 2.2. Synthesized 16-bit block.

```
entity LZD_BEH32 is
        port ( I :  in BIT_VECTOR (31 downto 0);
               O :  out INTEGER range 0 to 31;
               E :  out BIT  );
end LZD_BEH32;
architecture BEHAVE32 of LZD_BEH32 is
begin
        process (I)
          begin
          if (I(0) = '1') then
                O <= 0;   E <= '1';
          elsif (I(1 downto 0) = "10") then
                O <= 1;   E <= '1';
          elsif (I(2 downto 0) = "100") then;
                O <= 2;   E <= '1';
                .
                .
                .
          else      E <= '0';
          end if;
        end process;
end BEHAVE32;
```

Figure 3.1. Behavioral VHDL description for the LZD.

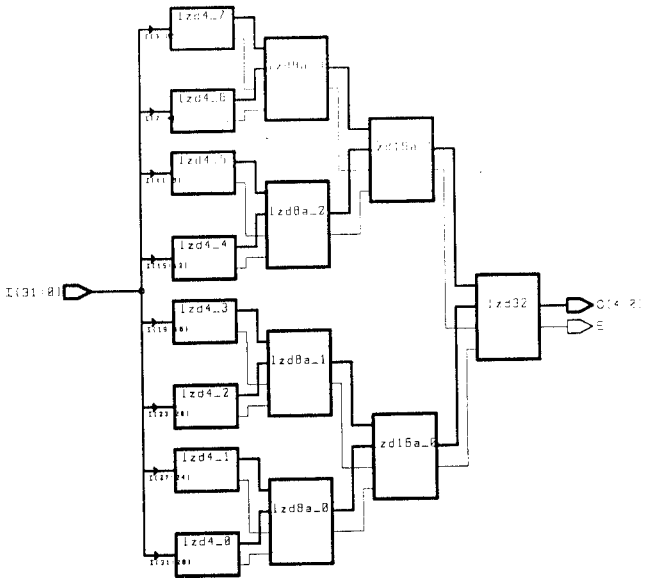

Figure 3.2. Hierarchical 32-bit LZD circuit.

Table 4.1. Nominal delay of the LZD circuits

VHDL	16-bit LZD	32-bit LZD	64-bit LZD
behavioral	2.75 ns	4.35 ns	6.50 ns
guided-structural	2.24 ns	2.73 ns	3.35 ns
Improvement	19 %	37 %	48 %

Table 4.2. Normalized Area [N units] of the LZD circuits

VHDL	16-bit LZD	32-bit LZD	64-bit LZD
behavioral	241	564	1308
guided-structural	208	426	861
Improvement	14 %	24 %	34 %

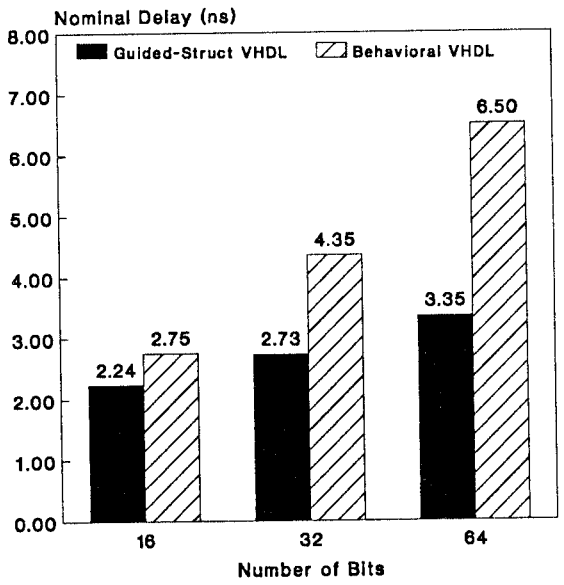

Figure 4.1. Speed comparison.

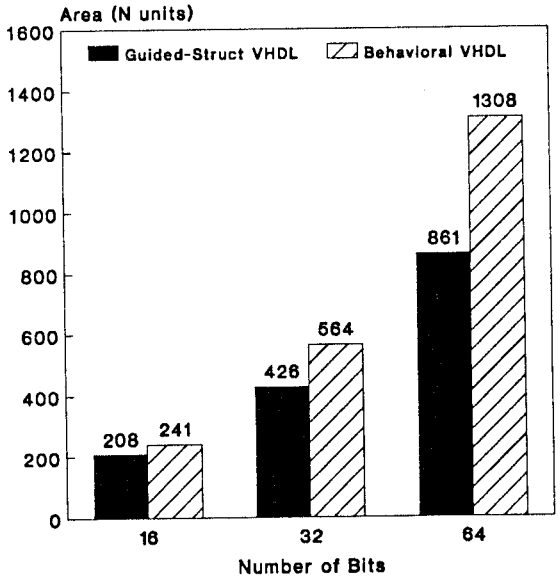

Figure 4.2. Area comparison.