Behavioral to Structural Translation in ESOP Form

M. A. Thornton, V. S. S. Nair
Department of Computer Science and Engineering

Southern Methodist University
Dallas, Texas 75205

Abstract

A translator for behavioral to structural descriptions of combinational logic circuits is presented. The input is in the form of a Boolean equation using Verilog syntax and the output is a Verilog net-list. The structure of the output circuit is in terms of an Exclusive-OR Sum-of-Products (ESOP) form which is noted for ease of testability and a reduced number of logic gates as compared to traditional Sum-of-Products forms. Also, since many FPGA devices are including the XOR gate as a basic structure, there is interest in ESOP circuit synthesis for FPGA implementations. The numerical methodology used to perform the translation is discussed in detail.

1 Introduction

Modern digital system design efforts typically involve the designer or engineer to initially specify the circuit to be implemented followed by (or in conjunction with) a description of the behavior of the circuit. Many times, the behavioral description of the circuit is documented through the use of a Hardware Description Language (HDL) such as *Verilog*. After these initial tasks are performed, there is an increasing trend toward the use of automated design tools for the remaining design tasks. In particular, the translation from the behavioral circuit description to a structural one is becoming more the domain of an automated tool rather than a manual translation task performed by the designer. In this paper, we present the results of a system that translates combinational logic expressions into an Exclusive-OR Sum Of Products (ESOP) structural form.

The choice for using the ESOP form has several advantages. It has been shown that ESOP forms generally require fewer logic gates than the more traditional Sum-Of-Products forms [1]. For symmetric functions it has been proven that the ESOP circuit form will never require more product terms than the number needed for the corresponding SOP realization [2]. Many FPGAs are now being manufactured that utilize the XOR gate as a basic cell component This requires ESOP implementations to be considered when an FPGA implementation is desired. Also, the ESOP form allows all single stuck-at faults to be tested with a minimal number of test vectors [3]. Recently, the method of utilizing the Reed-Muller transform to compute the Generalized Reed-Muller (GRM) circuit coefficients has been generalized so that all calculations may be performed using real-valued arithmetic [4] allowing for efficient numerical calculations and algorithms to be utilized in the synthesis process. Finally, recent developments in layout technology have reduced the layout area required for the XOR gate so that it is comparable with other basic logic gates [5].

The use of automatic behavioral to structural translators in the design of digital circuits allows the designer to spend more time in the specification stage of the design cycle. This is particularly advantageous when the desired circuit is specified with a behavioral description in terms of an HDL. By using automatic behavioral to structural translation tools, the design process becomes closer to the goal of simply specifying a circuit and then using tools to perform the implementation. Another advantage is that errors introduced during the manual translation of the behavioral to structural circuit description is eliminated allowing for faster design turn-around time and ultimately reducing the design cost.

This rest of the paper is organized as follows. Section 2 will provide a brief background of the mathematics involved in the synthesis of ESOP forms of combinational circuits. This will include a review of traditional techniques as well as new ideas that are used in this work. Following section 2, a discussion of the implementation of a behavioral to structural translation tool will be provided. This discussion will

0-8186-5655-7/94 $03.00 © 1994 IEEE

include implementation issues of the tool and its input and output considerations. Section 4 will provide experimental results of two circuits that have been translated by this tool. Finally, section 5 will present the conclusions of this work and a discussion of future efforts that will be expended in this area.

2 Mathematical Background of the Synthesis Technique

The use of spectral methods for the synthesis of digital circuits has been proposed from many sources [6] [7] [8]. In particular, the Reed-Muller (RM) transform has been used for the realization of GRM circuits and is described in detail in [9] [10]. The RM transform utilizes modulo-2 arithmetic to compute the GRM circuit coefficients. Recently, the RM transform technique has been generalized such that all calculations may be performed using real-valued arithmetic allowing standard linear algebra packages to be employed for the spectral calculations [4].

The ESOP form is a general form of a combinational logic circuit that includes the GRM forms as a subset. By utilizing generalized RM transform techniques, the user may compute any desired ESOP form. For a specific ESOP form, a coefficient matrix is computed and a linear system is solved providing the coefficients, r_i, in equation 1.

$$f(x) = r_0 g_0 \oplus r_1 g_1 \oplus \ldots \oplus r_n g_n \qquad (1)$$

Where each g_i is a literal, the logic value '1', or a product term consisting of complemented and/or uncomplemented input variables.

The matrix is formulated by using columns that are output vectors of the g_i terms that are desired for the circuit. Any product terms may be used as long as the resulting matrix is of full rank. This matrix will be referred to as the G matrix. For example, suppose that the following product terms are desired to be used:

$$\begin{aligned} g_0 &= 1 \\ g_1 &= x_0 \\ g_2 &= x_1 \\ g_3 &= x_0 \cdot x_1 \end{aligned}$$

The resulting coefficient matrix is:

$$G = \begin{array}{cccc} g_0 & g_1 & g_2 & g_3 \\ \end{array} \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array} \right]$$

This is in fact the RM transformation matrix for a 2-input polarity-0 GRM form, but in general, any ESOP form may be realized by constructing a matrix with the appropriate g_i product functions.

Once the matrix is formulated, it is desired to obtain a linear combination (in terms of the XOR operator) of the columns of the matrix that produce the output vector of the function to be synthesized. Mathematically stated, we wish to solve the linear system in equation 2:

$$G\underline{R} = \underline{F} \qquad (2)$$

Where $\underline{R}$ is the solution vector that contains the coefficients r_i in equation 1 that have values from the set $\{0, 1\}$, and $\underline{F}$ is the output vector of the function that is being synthesized.

As mentioned above, these calculations are carried out using real-valued arithmetic. This is accomplished by exploiting the fact that the two algebraic rings, $\Re$ and $\Re'$, have a homomorphic function of $h(x) = x(\text{mod}2)$. In fact these two rings also satisfy the idempotence property, hence, they are Boolean rings [11].

The definition of the ring, $\Re$, is:

$$\Re : \{0, 1, \oplus, \cdot\}$$

Where 0 and 1 denote the logic values of zero and one, the addition operator, $\oplus$, denotes the binary XOR operation, and the multiplicative operator, $\cdot$, denotes the binary AND function.

The ring, $\Re'$, is defined as:

$$\Re' : \{\emptyset, I, +, \times\}$$

Where:

$$\emptyset \boxminus \{0, \pm 2, \pm 4, \pm 6, \ldots, \pm 2j, \ldots\}$$

and,

$$I \equiv \{\pm 1, \pm 3, \pm 5, \ldots, \pm(2i+1), \ldots\}$$

Thus, $\emptyset$ and I are the sets of all even integers (including 0) and all odd integers (excluding 0), respectively. The two operators in the set, $\Re'$, are the additive operator, $+$, and the multiplicative operator, $\times$. These operators perform real addition and real multiplication, respectively.

Since there exists a homomorphic relationship between these two rings and since each equivalence class contains a single member, all calculations are performed using the operators and values of $\Re'$ and transformed using $h(x)$ to $\Re$ to yield the unique solution.

The solution is unique since the coefficient matrix, G, is required to be of full rank, hence, from linear system theory, a unique solution vector, $\underline{R}$ exists. Detailed derivations and proofs of these conjectures are given in [4].

3 Implementation

The mathematical basis for the synthesis of ESOP combinational logic circuits discussed in the previous section is used as a basis for the implementation of a behavioral to structural translation tool. This tool allows the designer to specify the logic equations of a system without having to manually translate the equations into a net-list form.

This tool is implemented in three main modules. The first module is written in *Verilog* and is a driver that evaluates a behavioral description of a logic function and creates the output vector. The output vector is then used as input to the synthesis portion of the tool. This portion is written in the C programming language and first computes the coefficient matrix, G, followed by invoking a linear equation solver using the matrix, G, and the input vector, $\underline{F}$ to compute the solution vector, $\underline{R}$. Once $\underline{R}$ is computed, each of its coefficients are used in the third portion of this tool to create a file that contains a *Verilog* net-list in ESOP form.

An overall block diagram of this tool is given in Figure 1.

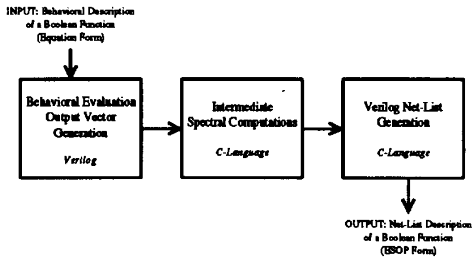
INPUT: Behavioral Description of a Boolean Function (Equation Form)
Behavioral Evaluation Output Vector Generation
Verilog
Intermediate Spectral Computations
C-Language
Verilog Net-List Generation
C-Language
OUTPUT: Net-List Description of a Boolean Function (ESOP Form)

Figure 1: Block Diagram of the Translation Tool

The translator is divided into these three components for various reasons. Chiefly, by separating the synthesis calculations, we could easily interchange other codes for this purpose. This will allow us to utilize the existing front-end for synthesis of functions that produce circuit types other than ESOP. In addition, the front-end is actually written in *Verilog* itself. The front-end of this tool evaluates a combinational logic module and produces the output vector of the particular function described by the module. This allowed us to obtain the $\underline{F}$ vector easily without parsing and evaluating the logic equation directly.

The second and third modules were implemented in the C programming language since the computations in these portions of the tool are of a numerical and string manipulation nature rather than simulating a circuit. The C library that contains "string" functions was especially handy for the back-end of the tool that produces the *Verilog* net-list output.

4 Experimental Results

This tool has successfully synthesized several different functions of varying complexity. Although it can be used to synthesize fairly large circuits, only two small circuits will be shown here for the sake of simplicity. There are many different ESOP forms of realization for a given logic function. However, for a specific set of of product terms (i.e., a set that can be used to formulate a coefficient matrix, G, of full rank), there is a single unique ESOP form. For this reason, there is no concept of minimization for a particular ESOP form although some ESOP forms are smaller than others. Finding the minimal ESOP form is currently an open problem and is beyond the scope of this paper.

The first circuit is a small 5-input function described by the following *Verilog* module:

Example 1:

```
module exmlbeh (f, inp1 , inp2 , inp3 ,
                inp4, inp5 );

 output f;
 input inp1,inp2,inp3,inp4,inp5;

 assign{f} = !(inp2 || (inp1 && inp3))
          || (!inp4) && inp1
          || inp5 && (!inp2);

endmodule
```

This behavioral description was used to generate the output vector for the synthesis computations. The output vector was generated by executing a *Verilog* driver module that looped 32 times and incremented a 5-bit register that served as input to the behavioral circuit module. At each iteration, the output of the circuit was written to a file.

The synthesis computations and net-list generator modules were invoked next. The input to these modules was the output vector file whose creation was discussed above. The resulting net-list file follows:

```
module rmcircuit (f, inp1 , inp2 ,
                  inp3 , inp4 , inp5 );

 output f;
 input inp1,inp2,inp3,inp4,inp5;

 and   g_0 ( wir0 , inp1, inp2);
 and   g_1 ( wir1 , inp1, inp3, inp4);
 and   g_2 ( wir2 , inp1, inp2, inp4);
 and   g_3 ( wir3 , inp1, inp2, inp3,
             inp4);
 and   g_4 ( wir4 , inp1, inp3, inp4,
             inp5);
 and   g_5 ( wir5 , inp1, inp2, inp3,
             inp4, inp5);
 xor   g_6 ( f, 1'b1,  inp2, wir0, wir1,
             wir2, wir3, wir4, wir5);
endmodule
```

□

The next example is a slightly larger circuit. The behavioral description for this circuit follows:

Example 2:

```
module exm2beh (f, inp1 , inp2 , inp3 ,
                inp4, inp5, inp6,
                inp7, inp8 );

 output f;
 input inp1,inp2,inp3,inp4,inp5,
       inp6,inp7,inp8;

 assign{f} = !(inp2 || (inp1 && inp3))
             || (!inp4) && inp1
             || inp5 && (!inp2)
             || inp3&&(!inp8
             || inp6&&inp7);

endmodule
```

The resulting output structural description is:

```
module rmcircuit (f, inp1 , inp2 , inp3 ,
                  inp4 , inp5 , inp6 ,
                  inp7 , inp8 );

 output f;
 input inp1,inp2,inp3,inp4,inp5,
       inp6,inp7,inp8;

and   g_0 ( wir0 , inp1, inp2);
and   g_1 ( wir1 , inp2, inp3);
and   g_2 ( wir2 , inp1, inp2, inp3);
and   g_3 ( wir3 , inp1, inp2, inp4);
and   g_4 ( wir4 , inp2, inp3, inp8);
and   g_5 ( wir5 , inp1, inp2, inp3,
            inp4);
and   g_6 ( wir6 , inp1, inp3, inp4,
            inp8);
and   g_7 ( wir7 , inp1, inp2, inp3,
            inp8);
and   g_8 ( wir8 , inp2, inp3, inp6,
            inp7, inp8);
and   g_9 ( wir9 , inp1, inp3, inp4,
            inp5, inp8);
and   g_10 ( wir10 , inp1, inp3,
             inp4, inp6, inp7, inp8);
and   g_11 ( wir11 , inp1, inp2, inp3,
             inp6, inp7, inp8);
and   g_12 ( wir12 , inp1, inp2, inp3,
             inp4, inp5, inp8);
and   g_13 ( wir13 , inp1, inp3, inp4,
             inp5, inp6, inp7, inp8);
and   g_14 ( wir14 , inp1, inp2, inp3,
             inp4, inp5, inp6, inp7,
             inp8);
xor   g_15 ( f, 1'b1,  inp2, wir0, wir1,
             wir2, wir3, wir4, wir5,
             wir6, wir7, wir8, wir9,
             wir10, wir11, wir12, wir13,
             wir14);
endmodule
```

□

These results show the input and output of our translation tool. The output form in these two examples is an ESOP form with all inputs uncomplemented. This is the polarity-0 GRM form. Other unique ESOP forms can also be computed by changing the coefficient matrix.

These examples contain a single logic function per behavioral module. We are currently implementing a partial *Verilog* parser using the UNIX utilities YACC and LEX. This will allow us to process modules with multiple logic equations in the near future.

5 Conclusions

We have presented the results of a behavioral to structural translator that utilizes the *Verilog* HDL for input and output. In addition, this tool is partially written in *Verilog* itself for the evaluation of input logic equations. The translator produces a net-list in ESOP form offering the many advantages present in this class of circuits such as ease of testability and a reduced logic gate count. The tool is constructed in a modular manner so that other synthesis functions may be used as they become available.

The mathematical methodology used as a basis for the implementation of the synthesis function in this tool has been presented and explained. Also, example circuits have been translated and were provided in the experimental results section of this paper.

Further work is planned to expand the capabilities of this tool. We are in the process of implementing other synthesis modules that will allow the user to specify the form of the structural circuit output. Another area of research is the investigation of pre-synthesis partitioning of the input circuits. This may result in the incorporation of partitioning methods proposed by other researchers [12], or, the incorporation of new partitioning techniques that are especially applicable for the specified structural output circuit type.

References

[1] Sasao, T. and Besslich, P., On the Complexity of Mod-2 Sum PLA's, *IEEE Trans. Comp.*, **vol. C-39**, pp.262-266, Feb. 1990.

[2] Rollwage, U., The Complexity of Mod-2 Sum PLA's for Symmetric Functions, *Proc. IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, Hamburg, Germany, pp. 6-12, Sept. 1993.

[3] Reddy, S.M. Easily Testable Realizations for Logic Functions, *IEEE Trans. Comp.* **vol. C-21, no. 11**, pp.1183-1188, 1972.

[4] Thornton, M. A. and Nair, V. S. S. A Numerical Method for Reed-Muller Circuit Synthesis, *Proc. IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, Hamburg, Germany, pp. 69-73, Sept. 1993.

[5] Sarabi, A. and Perkowski, M., Fast Exact Quasi-Minimal Minimization of Highly Testable Fixed-Polarity AND/XOR Canonical Networks, *Proc. DAC*, pp. 30-35, 1992.

[6] Hurst, S.L., Miller, D.M., and Muzio, J.C., **Spectral Techniques in Digital Logic**, Academic Press, Orlando, Florida, 1985.

[7] Edwards, C.R. The design of easily tested circuits using mapping and spectral techniques, *Radio and Elec. Eng.* **vol. 47, no. 7** (1977).

[8] Thornton, M. A. and Nair, V. S. S., An Iterative Combinational Logic Synthesis Technique Using Spectral Information, *Proc. Euro-DAC, European Design Automation Conference*, Hamburg, Germany, pp. 358-363, Sept. 1993.

[9] Green, D., **Modern Logic Design**, Addison-Wesley, Reading, Massachusetts, 1986.

[10] Davio, M., Deschamps, J.-P., and Thayse, A., **Discrete and Switching Functions**, Mc-Graw-Hill, New York, 1978.

[11] Bartee, T.C., Lebow, I.L., and Reed, I.S., **Theory and Design of Digital Machines**, Mc-Graw-Hill, New York, 1962.

[12] Hwang, T., Owens, R. M., and Irwin, M. J., Exploiting Communication Complexity for Multilevel Logic Synthesis, *IEEE Trans. on CAD*, **vol. 9, no. 10**, pp.1017-1027, Oct. 1990.