EASILY TESTABLE ARRAY MULTIPLIER DESIGN USING VHDL

S. M. Aziz
Department of Electrical and Electronic Eng.
Bangladesh University of Eng. & Technology
Dhaka-1000, Bangladesh
Fax: +880-2-863026
email: smaziz@eebuet.bdmail.net

Iftekhar Ahmed
Faculty of Engineering
University of Telecom Malaysia
75450 Bukit Beruang
Melaka, Malaysia
email: iftekhar@unitele.edu.my

ABSTRACT

Presents the design of variable array multipliers using VHDL. Multipliers of various operand sizes for different target processes can be implemented using the proposed VHDL based approach. The multipliers will be testable with a constant number of test vectors irrespective of the operand word lengths. A fast test pattern generator is also developed for simulation of the multiplier designs and subsequent testing of the fabricated chips.

Keywords: ***VHDL, Multiplier, C-testable, Parameterizable***

1.0 INTRODUCTION

With the continuing developments of VLSI technologies and tremendous shrinkage of process features, the need to develop process independent chip design tools are growing [1, 2]. The use of a hardware description language (HDL) for integrated circuit design eliminates the need to worry about process design rules at the early stages of design [2, 3]. This reduces the design complexity and time required to complete chip designs. This is very important since vendors need to market their products in the shortest possible time. The fact that such high level designs can be implemented on a variety of target processes reduces the design cost as well.

Multipliers are often one of the key elements in single chip digital information processors [4, 5]. The different modules within such processors have to be tried out for various architectures in order to find an optimal speed and area. Since it is time consuming to verify many possible layouts for each module, one approach is to use software packages called *module generators* or *silicon compilers* to provide fast and efficient design of parameterized modules. The multiplier compilers presented in [6] and [7] generate parameterizable layout for MOS technology. The technology independence of the compiler presented in [8] is limited by the requirement that the leaf cells have to be recharacterized in the new technology. The aim of this paper is to present the design of totally process independent VLSI array multipliers of variable size (parameterizable) using VHDL. The multipliers are made C-testable [9-13], i.e., they can be tested for all single stuck-at faults with a constant number of test vectors irrespective of the size of the operands. Although stuck-at fault models cannot adequately model transistor stuck-on and stuck-open faults [14, 15], it is possible to derive equivalent stuck-at test sets for logic gates to cover transistor stuck-on and stuck-open faults [16]. Since the number of test vectors is constant for any multiplier size, the test generation time for the proposed designs is considerably small.

In the next section the multiplier architecture is presented. Section 3 presents the testability of the multiplier. The VHDL model for the multiplier is given in Section 4.

2.0 ARCHITECTURE

The architecture of the multiplier is shown in Fig. 1. It is based on the modified Booth algorithm [12-13], [17-18]. This algorithm considers the operands as two's complement numbers and gives the product output in two's complement form. It reduces the number of partial products by almost a factor of two compared to the straightforward carry-save array multipliers [10, 11]. In the architecture of Fig. 1, the explicit sign extension circuitry has been eliminated by recoding the most significant bits (MSBs) of the partial products as a two's complement number [19, 20]. The multiplicand and the multiplier are denoted by X and Y respectively, where X = (x_5 x_4 x_3 x_2 x_1 x_0) and Y= (y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0) are chosen for convenience.

Although the architecture shown in Fig. 1 is that of a 6 by 8 bit multiplier, the regularity of the architecture enables one to draw larger multipliers quite easily. The multiplier of Fig. 1 contains six extra inputs x-1, y-1, e1, e2, e3, e4 for enhancing the testability of the multiplier. For normal multiplication operation these inputs should have the following logic levels: $x_{-1} = 0$, $y_{-1} = 0$, $e_1 = e_2 = 0$ and $e_3 = e_4 = 1$. The architecture of Fig. 1 is very regular consisting of only three different *leaf cells*, namely, the modified Booth encoder (BE), multiplexer-complementers and the adders. The final adders often employ some form of *fast* carry propagation scheme and are implemented differently than the carry-save adders (FA) used in the array [21].

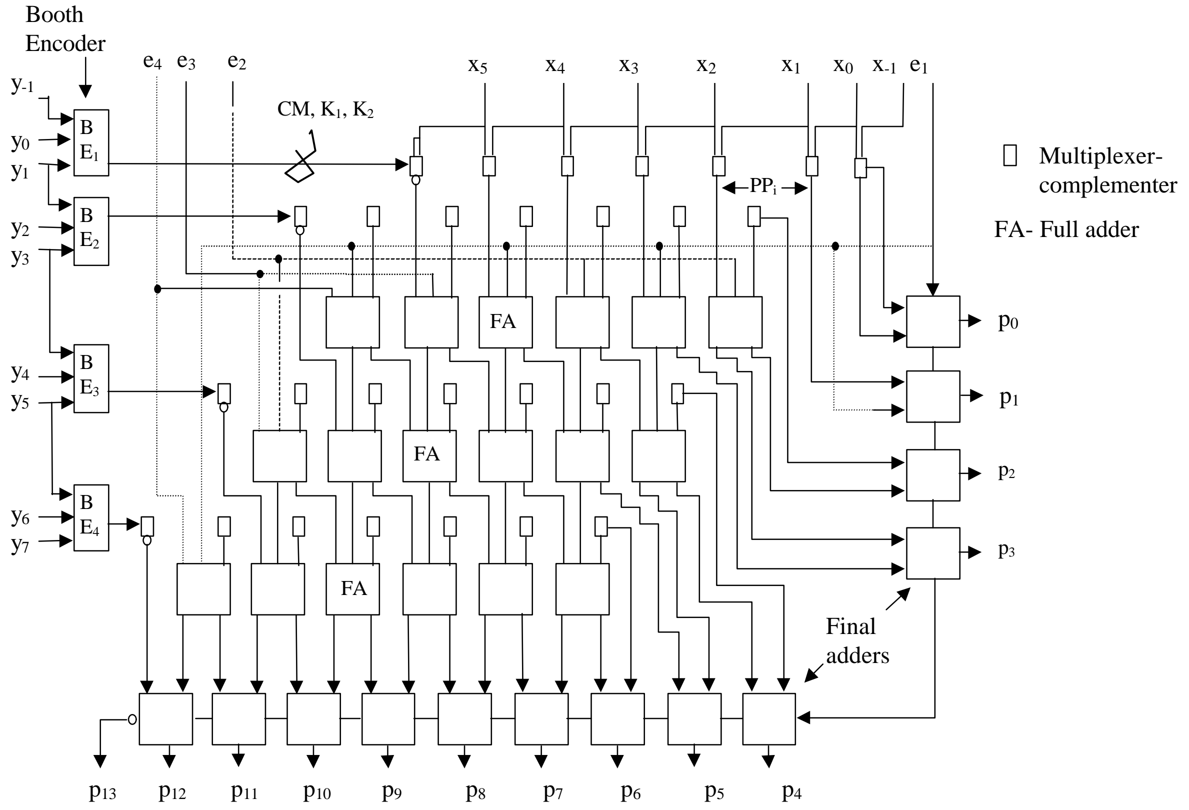

Fig. 1: Architecture of the multiplier with recoded sign bits
(Horizontal signals are omitted for clarity)

However, both these adders perform the same logic functions and therefore may be regarded to be the same from the point of view of logic functionality. Each modified Booth encoder scans three multiplier bits and generates three controls signals K_1, K_2 and CM. The multiplexer-complementers use these signals to generate the proper version of the multiplicand as a partial product (PP_i) according to the modified Booth recoding scheme [18]. The logic functions performed by the various cells of the multiplier of Fig. 1 are expressed by the following equations. Note that Z_i is the output of the multiplexer and PP_i is the partial product generated.

Modified Booth encoders:

$$K_1 = y_{i-1} \oplus y_i \tag{1}$$

$$K_2 = y_{i-1} y_i \overline{y}_{i+1} + \overline{y}_{i-1} \overline{y}_i y_{i+1} \tag{2}$$

$$CM = \overline{y}_{i-1} y_{i+1} + \overline{y}_i y_{i+1} \tag{3}$$

Multiplexer-complementer:

$$Z_i = K_1 x_i + K_2 x_{i-1} \tag{4}$$

$$PP_i = CM \oplus Z_i \tag{5}$$

Adders:

$$SUM = A \oplus B \oplus C \tag{6}$$

$$COUT = AB + BC + CA \tag{7}$$

3.0 TESTABILITY OF THE MULTIPLIER

Multipliers of any size based on the architecture presented in Fig. 1 can be made testable using a set of only 19 test vectors shown in Table 1. In this table, an 8-bit multiplicand (X) and an 8-bit multiplier (Y) are shown. The underlined bits denote the extensions required for larger multiplier [13]. For example, t_9 shows that X=0101 01<u>01</u> and Y=<u>1100</u> 1100. The vector X will be 01010101*0* and 0101010101*01* for 9-bit and 10-bit words respectively. Note that extensions are made to the right side. The vector Y will be *00*11001100 and *1100*11001100 for 10-bit and 12-bit words respectively. In this case extensions are made to the left side. All the adders are tested exhaustively using the vectors listed in Table 1. However, the modified Booth encoders and multiplexers are tested for stuck-at faults. This is because the complexity of exhaustively testing the later is enormous.

All the carry-save full adders in the multiplier of Fig. 1 are tested exhaustively by vectors t_1 - t_{12}. Exhaustive testing of

the final adders are accomplished by the vectors t_1 - t_{16}. Since the *sum* outputs of the adders are XOR function of the three inputs, and both the outputs of all the array full-adders are cascaded through a number of such XOR gates up to the final adders, the propagation of a faulty output in any of the adders to the primary outputs is guaranteed [13].

Table 1: Test vectors for the proposed multiplier

Test vectors	X	x_{-1}	Y	y_{-1}	$e_4e_3e_2e_1$
t_1	0000 0000*	0	0000 0000	0	0000
t_2	1000 0000	0	0101 0101	0	0000
t_3	1111 1111	1	0101 0101	0	1111
t_4	1000 0000	0	1010 1010	1	1111
t_5	0000 0000	0	1111 1111	0	1100
t_6	0000 0000	0	0011 0011	1	0011
t_7	1111 1111	1	0101 0011	0	0011
t_8	1111 1111	1	0100 0100	1	1100
t_9	0101 0101	0	1100 1100	1	0110
t_{10}	01010101	0	0011 0011	0	1001
t_{11}	1111 1111	1	0011 0011	0	0011
t_{12}	1111 1111	1	1100 1100	1	1100
t_{13}	0000 0000	0	1010 1010	1	1100
t_{14}	1111 1111	1	1010 1010	1	0000
t_{15}	111 11111	0	1001 1001	1	1111
t_{16}	111 11111	0	0110 0110	0	0000
t_{17}	0000 0000	0	0110 0110	0	1111
t_{18}	1111 1111	1	1001 1001	1	1001
t_{19}	1111 1111	1	1111 1111	1	0000

* The bits to be replicated for larger multipliers are underlined

Any single stuck-at fault in a modified Booth encoder is detected at it's outputs by the application of the set of inputs {001, 011, 100, 110, 111} to y_{i-1} y_i y_{i+1}. The test vectors t_{11}, t_{12}, t_{15}, t_{16} and t_{19} ensure the application of these patterns to all the encoders. These vectors also guarantee the propagation of the fault effect in an encoder to the primary outputs. The multiplexers require the set of patterns {0010, 0111, 1000, 1101} for the inputs $K_1x_iK_2x_{i-1}$ to test all single stuck-at faults. All the multiplexers in the multiplier of Fig. 1 receive these patterns by the application of the vectors t_2, t_3, t_5, t_{17} and t_{18}, and the effect of a fault is propagated to the primary outputs.

3.1 Test Generation for Variable Multipliers

The number of test vectors required is always 19, only their lengths vary with the multiplier size. Therefore, the test vectors for the proposed multipliers of variable operand word lengths can be generated using a simple program. A *C++ program* has been developed for test generation for variable multipliers. The pseudo-code for the test generation program is given below:

```
//Pseudo-code for the C-test generation program

#define T_Length  19

struct test_c {
   unsigned long X, Y;      // X, Y operands
   unsigned int Y_1, E;     // Y_1, test inputs E
};

void main()
{
   struct test_c T[19];
   int i, j, size_x, size_y;
   unsigned long v;
   unsigned int s;

   for (i=0; i<T_Length; i++) {
      T[i].X = 0;
      T[i].Y = 0;
      T[i].Y_1 = 0;
      T[i].E = 0;
   }

   // Test vector for X input
   for (i=0; i<T_Length; i++) {
         v = 0;
         switch (i) {
            case 1|3:  v = 1;
                       v <<= (size_x-1);
                  break;

            case 2|6|7|10|11|13|14|15|19:
                  for (j=0; j<size_x; j++) {
                       v <<= 1;
                       v |= 1;
                  }
                  break;

            case 8|9: for (j=0; j<(size_x/2); j++) {
                       v <<= 2;
                       v |= 1;
                  }
                  break;
            default: v=0; break;
         }
```

```
T[i].X = v;
}
// Test vector for Y input
// Actual index in paper is i+1
for (i=0; i<T_Length; i++) {
      v = 0;
      switch (i) {
         case 1|2|16: for (j=0; j<size_y/2; j++) {
                     v <<= 2;
                     v |= 1;
               }
               break;

         case 6:  for (j=0; j<size_y/2; j++) {
                     v <<= 2;
                     v |= 1;
               }
               v -= 2;
               break;

         case 4|19: for (j=0; j<size_y; j++) {    // all One
                     v <<= 1;
                     v |= 1;
               }
               break;

         case 3|12|13: for (j=0; j<size_y/2; j++) {
                     v <<= 2;
                     v |= 1;
               }
               v <<= 1;
               break;

         case 5|9|10: for (j=0; j<(size_y/4); j++) {
                     v =  (v << 4)| 3;
               }
               break;

         case 8|11: for (j=0; j<(size_y/4); j++) {
                     v =  (v << 4)| 3;
               }
               v <<= 2;
               break;

         case 7:  for (j=0; j<(size_y/4); j++) {
                     v = (v << 4) | 4;
               }
               break;

         case 14|18: for (j=0; j<(size_y/4); j++) {
                     v = (v << 4) | 9;
               }
               break;

         case 15|17: for (j=0; j<(size_y/4); j++) {
                     v =  (v << 4)| 3;
               }
               v <<= 1;
               break;
         default: v=0; break;
      }
      T[i].Y = v;
   }

   // Test vector Y_1
   for (i=0; i<T_Length; i++) {
      s = 0;
      switch (i) {
         case 3|5|7|8|11|12|13|14|18|19: s = 1; break;
         default: s=0; break;
      }
      T[i].Y_1 = s;
   }

   //Test vector test points E
   for (i=0; i<T_Length; i++) {
      s = 0;
      switch (i) {
         case 2|3|14|17: s =  15; break;
         case 4|7|11|12: s = 12; break;
         case 5|6|10: s = 3; break;
         case 9|18: s = 9; break;
         case 8:   s =  6; break;
         default: s=0; break;
      }
      T[i].E = s;
   }
}
```

The above program uses the *gettime* library function to calculate the test generation time. It can measure time up to millisecond range. However, when this program was used to generate test vectors for a 16 by 16-bit multiplier on a Pentium 166 MHz PC, it reported a test generation time of 0 seconds. This means that the total test generation time is less than a millisecond. However, another *C++ program* required approximately 5 minutes to generate exhaustive tests [10] for a 16 by 16-bit multiplier on the same PC. Therefore, the C-test generation scheme presented here is very economical in terms of the CPU time required.

4.0 VHDL MODELING

A VHDL model based on the architecture presented above has been developed for fast design and validation of process independent variable array multipliers. As mentioned before, three *basic cells* need to be modeled conforming to the logic functions represented by Equations 1 to 7. Generic modeling is required to deal with any

arbitrary operand word length in order to make the design parameterizable. The VHDL models of the modified Booth encoder and multiplexer-complementer are presented in the following sections. Also, the model for a row of adders of arbitrary length *n* in the multiplier array is presented using the generic VHDL functions in order to clarify the modeling process.

4.1 Modified Booth Encoder

```
entity Booth_encoder is
   port (y_i_1, y_i, y_i_2: in std_logic;
          K1, K2, CM: out std_logic
         );
end Booth_encoder;

architecture dataflow of Booth_encoder is
begin
   K1 <= y_i_1 xor y_i;
   K2 <= (y_i_1 and y_i and (not y_i_2)) or
          ((not y_i_1) and (not y_i) and y_i_2);
   CM <= ((not y_i_1) and y_i_2) or
          ((not y_i and y_i_2);
end dataflow;
```

4.2 Multiplexer-Complementer

```
entity mc is
   port (x_i, x_i_1, K1, K2, CM : in std_logic;
         Z_i, PP_i : out std_logic
        );
end mc;

architecture rtl of mc is
begin
          Z_i <= ((x_i and K1) or (x_i_1 and K2));
          PP_i <= CM xor Z_i;
end rtl;
```

4.3 Generic N Number Of 1-bit Full Adders

```
entity n_adder is
   generic (N : integer := 8);
   port  (A, B, Cin: in std_logic_vector (N-1 downto 0);
           Sum, Cout: out std_logic_vector (N-1 downto 0)
          );
end n_adder;

architecture rtl of n_adder is
begin
   process (A, B, Cin)
   begin
          Sum <= A xor B xor Cin;
          Cout <= (A and B) or (B and Cin) or (Cin and A);
   end process;
end rtl;
```

The overall multiplier has been coded hierarchically in VHDL using a number of generic modules of arbitrary size using the basic cells. The VHDL code has been validated by simulation at the behavioral level for various multiplier sizes using the test vectors generated by the *C++ program* in accordance with Table 1. Fig. 2 shows the results of one such simulation using Synopsis tools. The various values of X and Y shown in this figure are in hexadecimal. Note that Y is always kept equal to 02H for convenience. For X values of 82H and 06H, the product outputs are found to be FF04H and 000CH respectively. These are the expected product outputs for two's complement multiplication. After validation of the VHDL code at the behavioral level, a logic circuit was also synthesized.

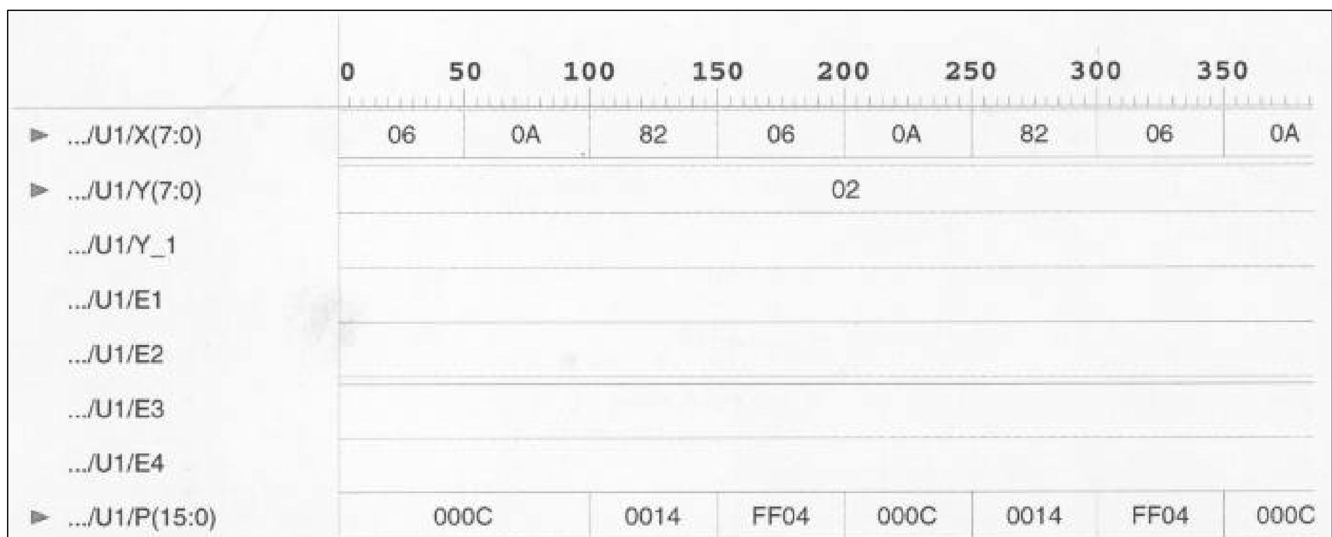

Fig. 2: Behavioral simulation results for an 8 by 8-bit multiplier

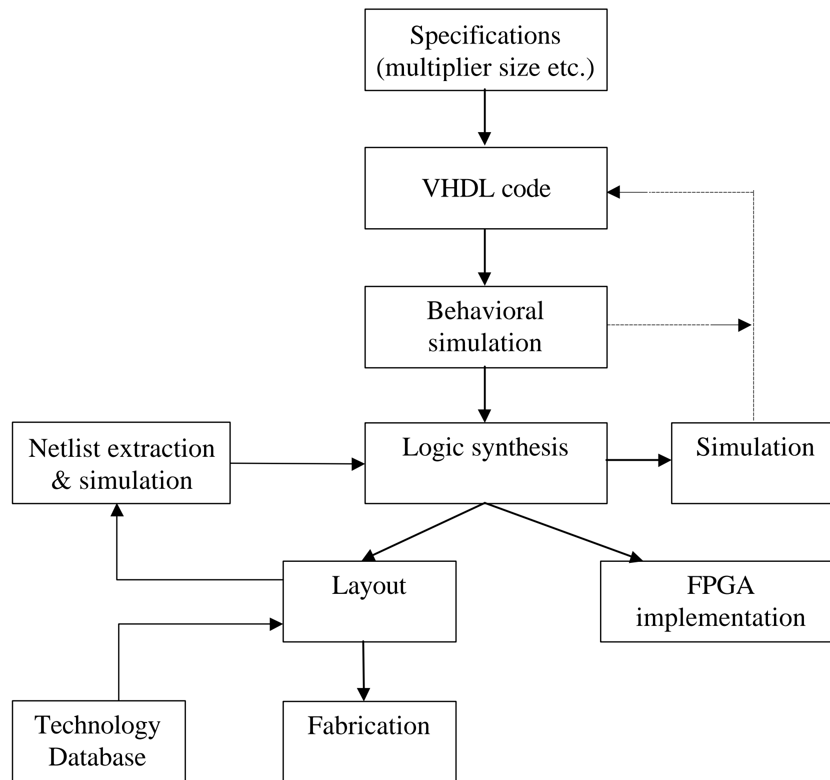

Fig. 3: Block diagram of the VHDL based design flow

The synthesized circuit can be checked for functionality, speed etc. using the same test vectors. Industry standard layouts (CIF or GDS-II) may be generated for the synthesized circuit using the appropriate technology database. The same VHDL code may be used to generate *bit-stream* to implement the design on Field Programmable Gate Arrays (FPGA) from commercial vendors. The block diagram of the overall process is shown in Fig. 3.

5.0 CONCLUSIONS

A VHDL based design of parameterizable array multipliers has been presented. The multipliers can be tested for all single stuck-at faults using only 19 test vectors. *A C++ program* generates the test vectors for the variable array multipliers within fraction of milliseconds. Since the multipliers are designed using VHDL, they can be ported from one process to another without any difficulty enabling fast fabrication of the designs. The VHDL code may also be used for FPGA implementation.

ACKNOWLEDGMENT

The authors acknowledge CAD facilities used at the Faculty of Computer Science and Information Technology, University of Malaya for this work.

REFERENCES

[1] D. D. Gazski, N. D. Dutt, C. H. Wu and Y. L. Lin, *High-Level Synthesis, Introduction to Chip and System Design*, Kluwer Academic Publishers, 1991.

[2] Z. Navabi, *VHDL Analysis and Modeling of Digital Systems,* New York, McGraw-Hill, Inc., 1993.

[3] J. R. Armstrong, *Chip Level Modeling with VHDL*, Englewood Cliffs, New Jersey, Prentice-Hall International, 1989.

[4] Frank P. J. M. Welton, Antoine Delaruelle, “A 2-µm CMOS 10-MHz Microprogrammable Signal Processing Core With an On-Chip Multiport Memory Bank”, *IEEE J. of Solid-State Circuits*, Vol. SC-20, No.3, June 1985, pp. 754-760.

[5] K. Takeda, F. Ishino, and Y. Ito et al., "A Single-Chip 80-bit Floating Point Processor", *IEEE J. of Solid-State Circuits,* Vol. SC-20, No. 5, October 1985, pp. 986-991.

[6] N. F. Benschop, "Layout Compilers for Variable Array Multipliers", in *Proc. of Custom Integrated Circuits Conf.*, May 1983, pp. 336-339.

[7] K. C. Chu and R. Sharma, "A Technology Independent MOS Multiplier Generator", in *21st Design Automation Conf.*, 1984, pp. 90-97.

[8] G. Venzl and R. Mitchell, "A Compilable Binary Tree Parallel Multiplier Designed for Speed and testability", in *Proc. of Custom Integrated Circuits Conf.*, 1989, pp. 93-94.

[9] A. D. Friedman, "Easily Testable Iterative Systems", *IEEE Trans. Comput.,* Vol. C-22, December 1973, pp. 1061-1064.

[10] J. P. Shen and F. J. Ferguson, "The Design of Easily Testable VLSI Array Multipliers", *IEEE Trans. Comput.,* Vol. C-33, No. 6, June 1984, pp. 554-560.

[11] A. R. Takach and N. K. Jha, "Easily Testable Gate-Level and DCVS Multipliers", *IEEE Trans. Computer-Aided Design*, Vol. 10, No. 7, July 1991, pp. 932-942.

[12] W. A. J. Waller and S. M. Aziz, "A C-Testable Parallel Multiplier Using Differential Cascode Voltage Switch (DCVS) Logic", in *Proc. of IFIP Int. Conf. on VLSI: VLSI'93*, Grenoble, France, September 6-10, 1993, pp. 3.4.1-3.4.10.

[13] S. M. Aziz, "A C-Testable Modified Booth Array Multiplier", in *Proc. of Int. Conf. VLSI Design*, New Delhi, India, January 4-7, 1995, pp. 278-282.

[14] Y. K. Malaiya, "Testing Stuck-On Faults in CMOS Integrated Circuits", in *Proc. of Int. Conf. on CAD*, Santa Clara, CA, November 1984, pp. 248-250.

[15] S. M. Reddy and M. K. Reddy, "Testable Realizations for FET Stuck-Open Faults in CMOS Combinational Logic Circuits", *IEEE Trans. Comput.,* Vol. C-35, No. 8, August 1986, pp. 742-754.

[16] S. M. Reddy, V. D. Agrawal and S. K. Jain, "A Gate Level Model for CMOS Combinational Logic Circuits with Applications to Fault Detection", in *21st Design Automation Conf.*, 1984, pp. 504-509.

[17] S. M. Aziz, Z. Rouf and J. Kamruzzaman, "Signed and Unsigned Multiplication Using a Single Cellular Array", in *Proc. of 7th Int. Symp. on IC Technology, Systems and Applications*, Singapore, September 10-12, 1997, pp. 569-572.

[18] J. J. F. Cavanagh, *Digital Computer Arithmetic Design and Implementation*, New York, McGraw-Hill, Inc., 1985.

[19] M. Roorda, "Method to Reduce the Sign Bit Extension in a Multiplier That Uses the Modified Booth Algorithm", *Electronic Letters,* Vol. 22, No. 20, 25th September 1986, pp. 1061-1062.

[20] N. Burgess, "Removal of Sign-Extension Circuitry from Booth's Algorithm Multiplier-Accumulators", *Electronic Letters,* Vol. 26, No. 17, 16th August 1990, pp. 1413-1415.

[21] Neil H. E. Weste and Kamran Eshraghian, "Principles of CMOS VLSI Design - A Systems Perspective", USA, Addison-Wesley Publishing Company, 1993.

BIOGRAPHY

S. M. Aziz is a Professor of Electrical and Electronic Engineering at Bangladesh University of Engineering and Technology (BUET). He received B.Sc. and M.Sc. degrees in Electrical and Electronic Engineering from the same university in 1984 and 1986 respectively. He obtained his Ph.D. degree in Electronic Engineering in 1993 from the University of Kent, UK and specialized in VLSI Design and testability. In 1996, he was a visiting scholar at the University of Texas at Austin and worked at Crystal Semiconductor Corporation in advanced VLSI Circuit Design. His research interests include design for testability, VHDL based design, computer arithmetic and architectures, neural networks etc. He is a member of the *IEEE Computer Society*, *Circuits and Systems Society*, and *Solid State Circuits Society*.

Iftekhar Ahmed received B.Sc. degree in Electrical and Electronic Engineering from Bangladesh University of Engineering and Technology in 1986. He obtained his M.Sc. and Ph.D. degrees in Electrical, Electronic and Systems Engineering from University Kebangsaan Malaysia in 1990 and 1995 respectively. He served as a Lecturer in the Faculty of Computer Science and Information Technology of the University of Malaya from April 1995 to April 1998. Currently, he is a Lecturer in the Faculty of Engineering of the University of Telecom Malaysia. His research interests are in the areas of Digital System Design and Testing.