Engineering Letters, 13:3, EL_13_3_2 (Advance online publication: 4 November 2006)

An ASIC Architecture for Generating Optimum Mixed Polarity Reed-Muller Expression

Mozammel H. A. Khan
Department of Computer Science and Engineering, East West University, 43 Mohakhali, Dhaka 1212, BANGLADESH.
mhakhan@ewubd.edu

Abstract: Logic function realization using Reed-Muller (RM) expression has manifold advantages over realization using SOP expression. In this paper, we present an ASIC architecture for generating optimum 3-variable mixed polarity RM expression. The ASIC based minimization is much faster than the software based minimization. The design is modeled using Verilog HDL and synthesized using Quartus II 4.2 software for Stratix EP1S10F484C5 FPGA device.

Keywords: ASIC, FPGA based circuit synthesis, Logic synthesis, Reed-Muller expression, Verilog HDL

1. Introduction

Classically logic functions are expressed and minimized as sum of products (SOP) expressions resulting into AND-OR circuits. Alternative representation that found a widespread popularity is the Reed-Muller (RM) expressions resulting into AND-EXOR circuits. RM expressions have many advantages over classical SOP expressions. For many functions, AND-EXOR circuits require less AND gates than AND-OR circuits [9]. AND-EXOR circuits are also more easily testable than AND-OR circuits [4, 10]. Other very important uses of RM expressions are in the classification of Boolean functions and Boolean matching [13, 14]. AND-EXOR circuits have been used in arithmetic, error correcting, and telecommunication applications [11, 12].

There are seven classes of RM expressions [10]. Among them, fixed polarity RM (FPRM) expression and mixed polarity RM (MPRM) expression are very popular. MPRM expression is also known as Kronecker (KRO) expression [10]. Mixed polarity RM (MPRM) expression requires, in general, lesser products that FPRM expression. On the other hand, minimization of MPRM expression is more complex than FPRM expression. Almost all methods for minimization of FPRM and MPRM expressions are software based [2-8]. These software-based methods use sequential processing and require exponential computation time. So far our knowledge is concerned, only one ASIC based method is available for minimization of 4-variable fixed polarity RM (FPRM) expression [1].

In this paper, we present an ASIC architecture for minimization of 3-variable MPRM expressions and its FPGA based implementation. This architecture can be extended for larger variables. ASIC-based method is inherently parallel in nature and requires constant time.

2. Background

RM expressions are formed by successive applications of the following expansions.

Positive Davio (pD) expansion:

$$f(x_1,\cdots,x_i,\cdots,x_n) = f_0 \oplus x_i f_2$$

Negative Davio (nD) expansion:

$$f(x_1,\cdots,x_i,\cdots,x_n) = f_1 \oplus x_i' f_2$$

Shannon (S) expansion:

$$f(x_1,\cdots,x_i,\cdots,x_n) = x_i' f_0 \oplus x_i f_1$$

where,

$$f_0 = f(x_1,\cdots,x_{i-1},0,x_{i+1},\cdots,x_n),$$
$$f_1 = f(x_1,\cdots,x_{i-1},1,x_{i+1},\cdots,x_n),$$
$$f_2 = f_1 \oplus f_2.$$

If we apply the pD expansion on a variable, then that variable appears only in uncomplemented form (polarity 0) throughout the expression. If we apply the nD expansion on a variable, then that variable appears only in complemented form (polarity 1) throughout the expression. If we apply the S expansion on a variable, then that variable appears in both complemented and uncomplemented forms (polarity 2) in the expression.

If we apply the pD expansion on all the variables of a function, then the resulting expression is called positive polarity Reed-Muller (PPRM) expression. The form of a 3-variable PPRM expression is

$$f(x_1,x_2,x_3) = a_{000} \oplus a_{001}x_3 \oplus a_{010}x_2 \oplus a_{011}x_2x_3 \oplus a_{100}x_1 \oplus a_{101}x_1x_3 \oplus a_{110}x_1x_2 \oplus a_{111}x_1x_2x_3$$

where, $(\forall i)a_i \in \{0,1\}$ are the PPRM coefficients.

If we apply the nD expansion on all the variables of a function, then the resulting expression is called negative polarity Reed-Muller (NPRM) expression. The form of a 3-variable NPRM expression is

$$f(x_1,x_2,x_3) = a_{000} \oplus a_{001}x_3' \oplus a_{010}x_2' \oplus a_{011}x_2'x_3' \oplus a_{100}x_1' \oplus a_{101}x_1'x_3' \oplus a_{110}x_1'x_2' \oplus a_{111}x_1'x_2'x_3'$$

where, $(\forall i)a_i \in \{0,1\}$ are the NPRM coefficients.

If we apply the S expansion on all the variables of a function, then the resulting expression is called canonical exclusive-OR sum of products (CESOP) expression. The form of a 3-variable CESOP expression is

$$f(x_1,x_2,x_3) = a_{000}x_1'x_2'x_3' \oplus a_{001}x_1'x_2'x_3 \oplus a_{010}x_1'x_2x_3' \oplus a_{011}x_1'x_2x_3 \oplus a_{100}x_1x_2'x_3' \oplus a_{101}x_1x_2'x_3 \oplus a_{110}x_1x_2x_3' \oplus a_{111}x_1x_2x_3$$

where, $(\forall i)a_i \in \{0,1\}$ are the CESOP coefficients. The CESOP expression is similar to the canonical sum of products expressions with OR operators replaced by EXOR operators.

If we apply a combination of pD and nD expansions on the variables of a function, then the resulting expression is called the fixed polarity Reed-Muller (FPRM) expression. If the polarity vector of a 3-variable function is $p = (010)$, then the form of the corresponding FPRM expression is

$$f(x_1,x_2,x_3) = a_{000} \oplus a_{001}x_3 \oplus a_{010}x_2' \oplus a_{011}x_2'x_3 \oplus a_{100}x_1 \oplus a_{101}x_1x_3 \oplus a_{110}x_1x_2' \oplus a_{111}x_1x_2'x_3$$

where, $(\forall i)a_i \in \{0,1\}$ are the FPRM coefficients. There are 2^n different polarity vectors for *n* variables and every polarity vector produces a different FPRM expression with different cost (number of non-zero coefficients). Therefore, the minimization of FPRM expression corresponds to finding a polarity vector $P \in \{0,1\}^n$ and the corresponding FPRM expression so that the cost is the minimum. There are several software based [2-5, 7] and one ASIC based [1] method for FPRM minimization.

If we apply a combination of pD, nD, and S expansions on the variables of a function, then the resulting expression is called the mixed polarity Reed-Muller (MPRM) expression. This expression is also known as Kronecker (KRO) expression [10]. If the polarity vector of a 3-variable function is $P = (012)$, then the form of the corresponding MPRM expression is

$$f(x_1,x_2,x_3) = a_{000}x_3' \oplus a_{001}x_3 \oplus a_{010}x_2'x_3' \oplus a_{011}x_2'x_3 \oplus a_{100}x_1x_3' \oplus a_{101}x_1x_3 \oplus a_{110}x_1x_2'x_3' \oplus a_{111}x_1x_2'x_3$$

where, $(\forall i)a_i \in \{0,1\}$ are the MPRM coefficients. There are 3^n different polarity vectors for *n* variables and every polarity vector produces a different MPRM expression with different cost. Therefore, the minimization of MPRM expression corresponds to finding a polarity vector $P \in \{0,1,2\}^n$ and the corresponding MPRM expression so that the cost is the minimum. There are several software based [5-8] methods for MPRM minimization. In this paper, we present an ASIC based method for MPRM minimization.

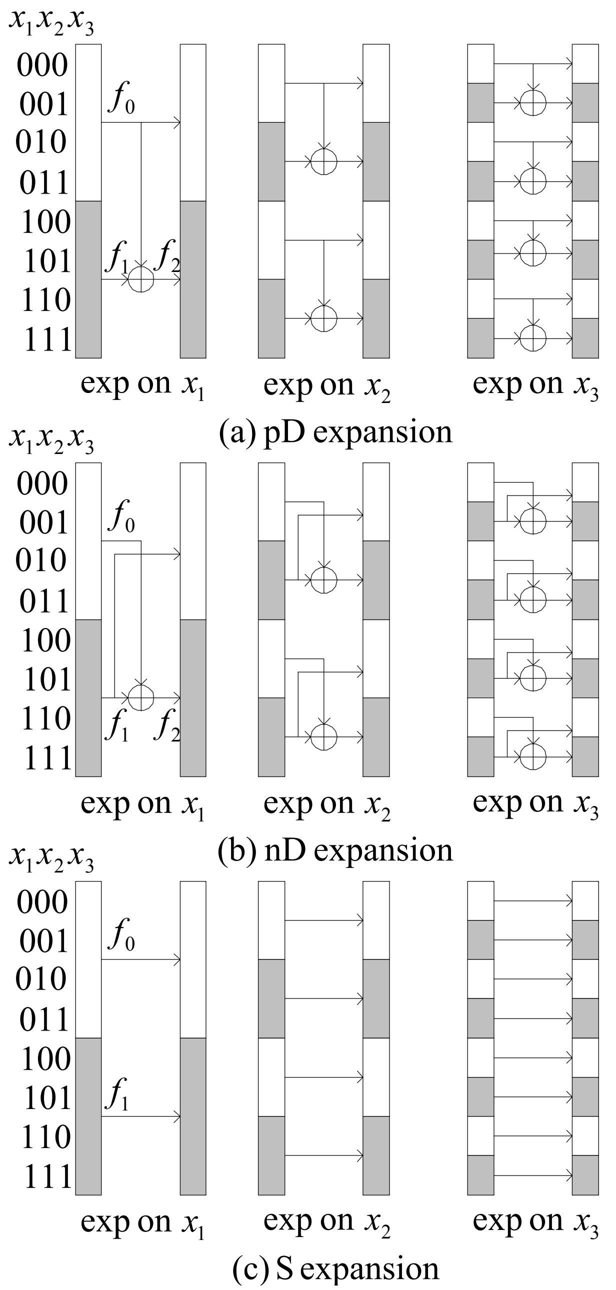


Figure 1. Computation of expansion coefficients on truth vector of a 3-variable function.

The cost of different forms of RM expressions are related as below:

$$PPRM, NPRM \geq FPRM \geq MPRM$$

Computation of expansion coefficients on truth vector of a 3-variable function is shown in Figure 1. If we apply pD expansion on x_1 (Figure 1a), then a_0 to a_3 remain unchanged and a_4 to a_7 are the bit-wise EXOR of a_0 to

a_3 and a_4 to a_7. Application of pD expansion on x_2 and x_3 can be explained in a similar manner. If we apply nD expansion on x_1 (Figure 1b), then a_4 to a_7 are copied to vector location 0 to 3 and the bit-wise EXOR of a_0 to a_3 and a_4 to a_7 are copied to vector location 4 to 7. Application of nD expansion on x_2 and x_3 can be explained in a similar manner. If we apply S expansion on either x_1, x_2, or x_3 (Figure 1c), the truth vector remain unchanged.

The MPRM coefficients for all polarity vectors for the 3-variable function $f(x_1, x_2, x_3) = [11100101]$ (expressed in truth vector) are shown in Table 1. The bold entries are the minimum solution and our developed system will find the first one (polarity 102) as the final solution.

3. The RTL Circuit

The RTL circuit for generating optimum 3-variable MPRM coefficients from Boolean coefficients (truth vector) is shown in Figure 2 and the detailed logic circuit of **convert** unit is shown in Figure 3.

All possible polarity vectors of a 3-variable function are generated by the **PCtr** counter (Figure 2). It is a 3-digit ternary counter with binary encoded output cp[5:0]. It counts from ternary 000 to 222 and the outputs are encoded as 000000 to 101010, respectively. cp[5:4] is the polarity for x_1, cp[3:2] is the polarity for x_2, and cp[1:0] is the polarity for x_3. Initially it is cleared to 000000 at the negative edge of the **Start** input. Then the value of the counter is incremented at the positive edge of every fourth clock enabled by the **en** input, which is 1 when c[1:0] = 11, that is, the last step of the processing cycle of generating MPRM coefficients for a given polarity vector.

Generating MPRM coefficients for a given polarity vector requires four steps. These steps are determined by the **SCtr** counter (Figure 2). The output c[1:0] determines the steps. It is initially cleared to 00 at the negative edge of the **Start** input and then incremented at the positive edge of every clock pulse.

The current polarity vector is stored in the register **CPReg** (Figure 2). The current polarity value cp[5:0] is copied to tp[5:0] at the positive edge of every fourth clock pulse enabled by the **en** input, which is 1 when c[1:0] = 11, that is, the last step of the processing cycle of generating MPRM coefficients for a given polarity vector.

Table 1. MPRM coefficients for all polarity vectors for the function $f(x_1, x_2, x_3) = [11100101]$

Polarity	MPRM coeffi r0 … r7	Cost	Polarity	MPRM coeffi r0 … r7	Cost
000	10011101	5	120	01011010	4
001	10110111	6	121	11110110	6
002	11011001	5	122	01011011	5
010	11011001	5	**200**	**10010100**	**3**
011	01111011	6	201	10111000	4
012	10011101	5	202	11010100	4
020	10111110	6	210	11010100	4
021	10010110	4	**211**	**01101000**	**3**
022	10010110	4	**212**	**10010100**	**3**
100	01001101	4	220	10110101	5
101	11000111	5	221	10011010	4
102	**01001001**	**3**	222	11100101	5
110	**01001001**	**3**			
111	11001011	5			
112	01001101	4			

The polarity vector producing the so far minimum cost expression is stored in register **PReg** (Figure 2). If the cost of the current MPRM coefficients (cs[3:0]) is less than the cost of the so far minimum cost (s[3:0]), then lt = 1 and the current polarity tp[5:0] is copied to p[5:0] at the positive edge of the clock pulse.

After every fourth clock pulse, the **convert** unit generates the MPRM coefficients for the current polarity vector. These coefficients are stored in the register **CRReg** (Figure 2). This register is initially preset to 11111111 so that the **cmp** unit does not generate a false lt = 1 at the beginning of the processing cycle due to undefined values of cr[0:7]. The values of tr[0:7] are copied to cr[0:7] at the positive edge of the first clock pulse of the next processing cycle enabled by **en** input, which is 1 when c[1:0] = 00.

The **adder** unit (Figure 2) determines the cost (cs[3:0]) of the current MPRM coefficient.

The register **SReg** (Figure 2) holds the cost of the so far minimum cost s[3:0]. It is initially preset to 1000 (2^3), the maximum possible cost.

The **cmp** unit (Figure 2) compares the value of the so far minimum cost (s[3:0]) and the current cost (cs[3:0]) and generates lt = 1 if cs[3:0] < s[3:0], which is then used to load the corresponding latest values to **PReg**, **SReg**, and **RReg**.

The so far minimum MPRM coefficients are stored in register **RReg** (Figure 2).

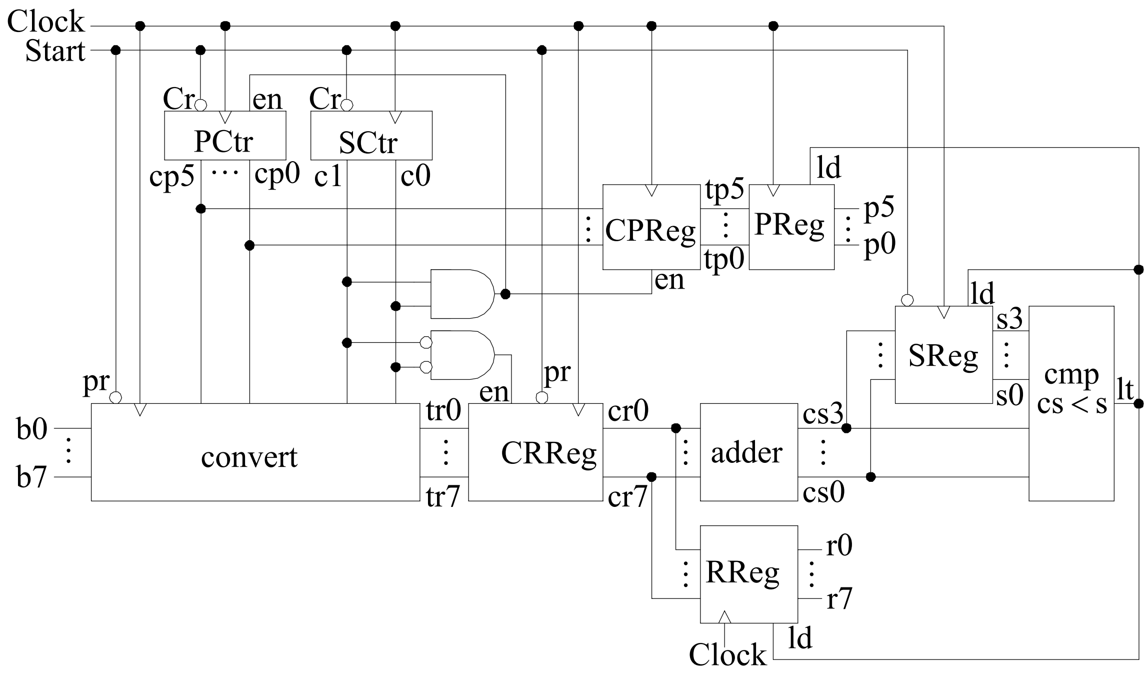


Figure 2. The RTL circuit for generating optimum 3-variable MPRM coefficients.

The detailed logic circuit of the **convert** unit is shown in Figure 3. The Boolean coefficients (truth vector) b[0:7] is input to the register **bReg** at the positive edge of each clock pulse. The register **TRReg** holds the intermediate values of the expansion. It is initially preset to 11111111 at the negative edge of the **Start** input so that the register **CRReg** (Figure 2) does not get any undefined value and generates a false lt = 1. The step count c[1:0] selects one of the four sets of expansion values through the column of 4×1 MUXs. The other 2×1 MUXs determine the expansion values as discussed in Figure 1. The left column of 2×1 MUXs generates the expansion on x_1, the middle column generates the expansion on x_2, and the right column generates the expansion on x_3.

At the positive edge of the first clock of the processing cycle, when c[1:0] = 00, b[0:7] are copied to tr[0:7].

If cp[5:4] = 00 (polarity 0), then tr[0:3] are copied to the location 0 to 3 and tr[0:3] ⊕ tr[4:7] are copied to location 4 to 7, which is pD expansion. These values are copied to register **TRReg** at the positive edge of the second clock pulse of the processing cycle when c[1:0] = 01. If cp[5:4] = 01 (polarity 1), tr[4:7] are copied to the location 0 to 3 and tr[0:3] ⊕ tr[4:7] are copied to location 4 to 7, which is nD expansion. If cp[5:4] = 10 (polarity 2), tr[0:7] are copied unchanged, which is S expansion.

cp[3:2] produces expansion on x_2 and that is copied to register **TRReg** at the positive edge of the third clock pulse of the processing cycle when c[1:0] = 10.

cp[1:0] produces expansion on x_3 and that is copied to register **TRReg** at the positive edge of the fourth clock pulse of the processing cycle when c[1:0] = 11.

The pipeline of the processing cycles is shown in Figure 4. When the **Start** input goes from 1 to 0, the register **TRReg** is preset to 11111111, **CRReg** to 11111111, **SReg** to 1000; and counter **PCtr** is cleared to 000000 and **SCtr** to 00. At clock 0, b[0:7] are loaded to register **bReg** and the circuit becomes ready for the processing. b[0:7] are copied to **bReg** register at every clock. This will make no harm in the processing, but the register design will be easier. At clock 1, c[1:0] = 00 and the **bReg** register is copied to the **TRReg** register through the column of 4×1 MUXs. At clock 2, c[1:0] = 01 and the expansion on x_1, as determined by the left column of 2×1 MUXs, is loaded to the register **TRReg** through the column of 4×1 MUXs. At clock 3, c[1:0] = 10 and the expansion on x_2, as determined by the middle column of 2×1 MUXs, is loaded to the register **TRReg** through the column of 4×1 MUXs. At clock 4, c[1:0] = 11 and the expansion on x_3, as determined by the right column of 2×1 MUXs, is loaded to the register **TRReg** through the column of 4×1 MUXs. At the same time cp[3:0] are also copied to the register **CPReg**. This ends the generation of MPRM coefficients for cp[5:0] = 000000. This clock also increments the counter cp[5:0] to 000001. From clock 5, a new process of generating MPRM coefficients for cp[5:0] = 000001 starts. At clock 5, the

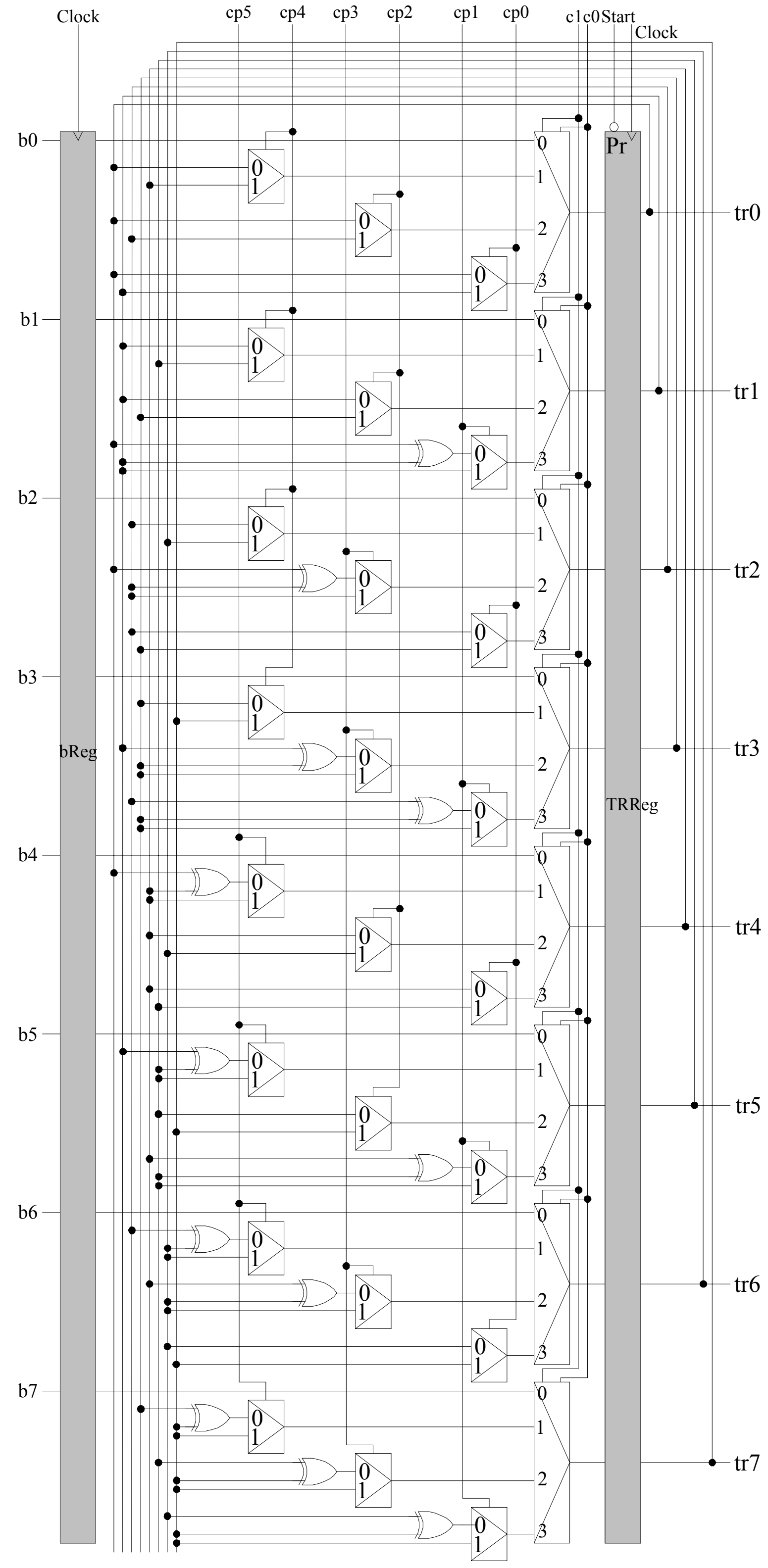


Figure 3. Detailed logic circuit of the convert unit of Figure 2.

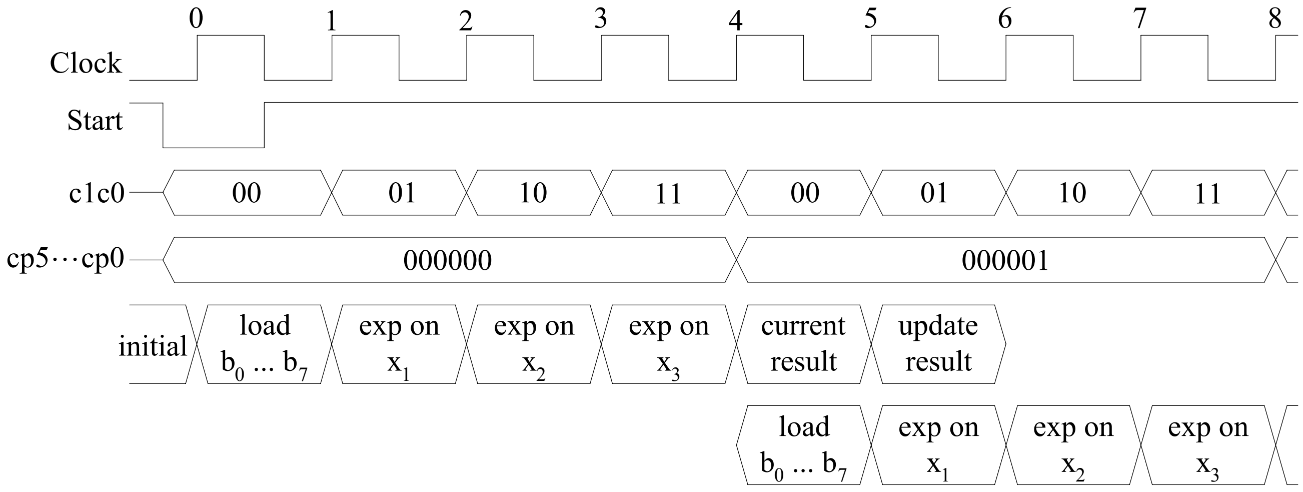


Figure 4. Part of the processing pipeline.

MPRM coefficient for cp[5:0] = 000000 (tr[0:7]) is stored in the register **CRReg**. At this point the **adder** unit calculates the cost of the coefficients (cs[3:0]) and compares that with the so far minimum cost (s[3:0]). If cs[3:0] < s[3:0], then lt = 1 and cr[0:7] are copied to the register **RReg** at clock 6. At the same time, the current polarity vector tp[5:0] is also copied to the register **PReg**. In this way the processing pipeline goes on and finally the minimum MPRM coefficients and the corresponding polarity vector are available at r[0:7] and p[5:0] outputs, respectively.

4. FPGA Implementation

The RTL circuit for generating optimum MPRM coefficients as described in Figure 2 and 3 is modeled using Verilog HDL and synthesized using Altera's **Quartus II 4.2** software for **Stratix EP1S10F484C5** FPGA device. The fitter required 90 logic elements and 24 pins. The timing analyzer reports worst-case tsu: 2.752 ns, worst-case tco: 7.558 ns, worst-case th: -1.940 ns, and clock setup: 162.10 MHz (period = 6.169 ns). The circuit is simulated with 162.10 MHz clock. Part of the functional simulation is shown in Figure 5 and part of the timing simulation is shown in Figure 6.

5. Conclision

Realization of logic circuits using Reed-Muller (RM) expressions (AND-EXOR expression) is advantageous over realization using SOP expression (AND-OR expression) in many aspects including requirement of lesser number of gates [9], easy testability of the circuit [4,10], ease of computation, etc. RM circuits also have applications in arithmetic, error correcting, and telecommunication circuits [11,12].

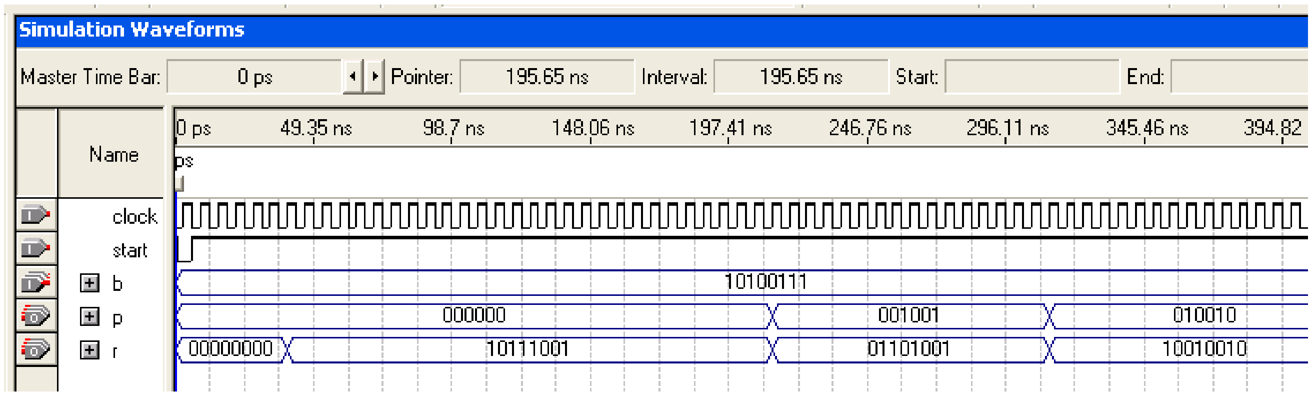
Boolean function classification and matching are very important in standard-cell based VLSI design. Boolean function classification and matching can be done using RM expression as tools [13,14]

Traditional software based methods of minimization of RM expressions require exponential computation time. This computation time can be drastically minimized if ASICs can be used to compute the minimized RM expression. Only one such attempt was made to minimize 4-variable fixed polarity RM (FPRM) expression in [1]. In this paper we presented an ASIC architecture for minimization of 3-variable mixed polarity RM (MPRM) expression. The initial attempt is to device architecture for the ASICS to minimize the MPRM expression and we worked with 3-variable case as a test case. The architecture can be extended for functions of more than 3 variables.

MPRM expressions are more general than FPRM expressions and require lesser products than FPRM expressions. On the other hand, minimization of MPRM expressions is more difficult than minimization of FPRM expressions. In this respect having an ASIC for minimization of MPRM expression is very important, because ASIC based minimization is inherently parallel and require constant time.

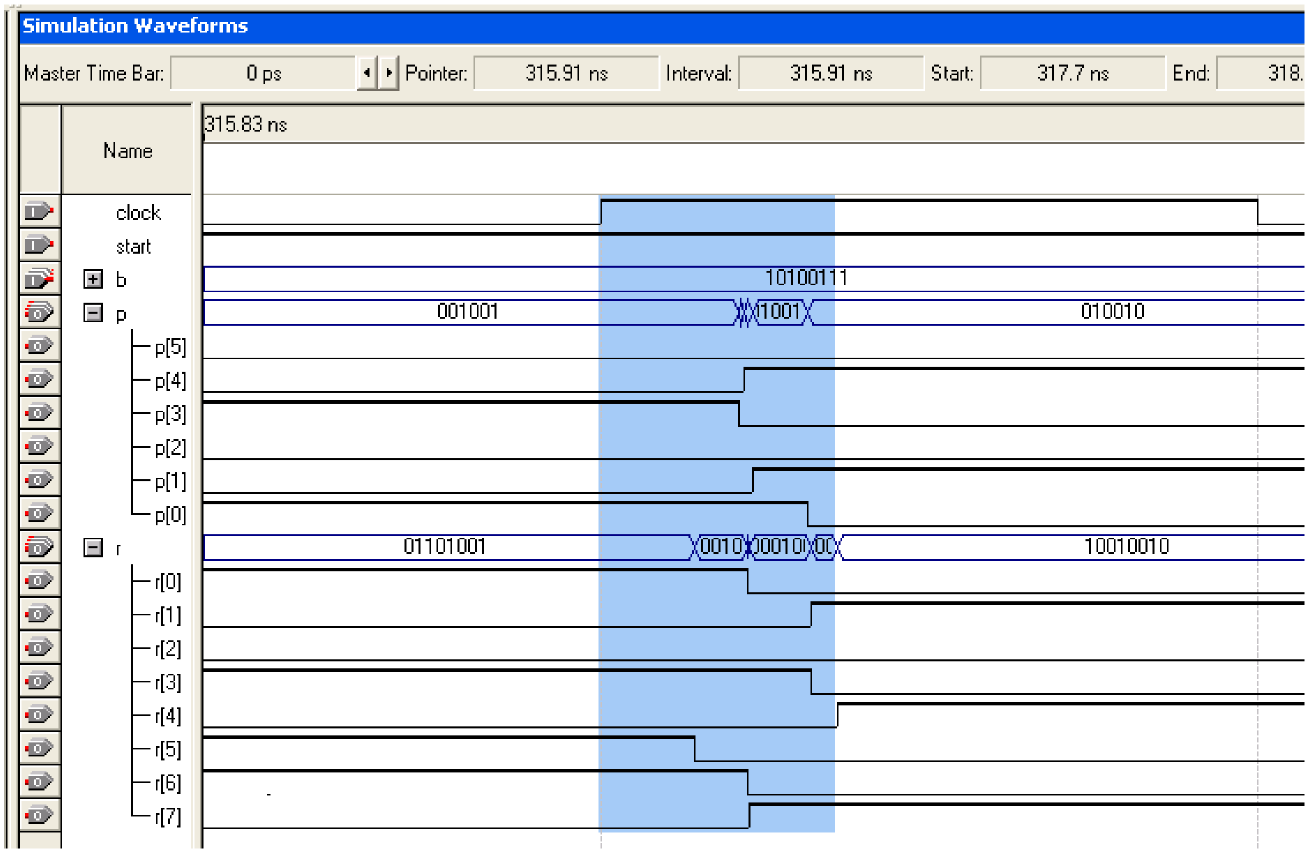
The RTL logic architecture for generating minimum MPRM expression is very easy to conceptualize and implement. The 3-variable circuit is modeled using Verilog HDL and synthesized using Quartus II 4.2 software for Stratix EP1S10F484C5 FPGA device, which requires only 90 logic elements.

The architecture of the circuit is highly modular and can be very easily extended for functions having more variables. Our future work focuses on developing parameterized Verilog model of the circuit so that it can be implemented for any number of variables.


Simulation Waveforms

Master Time Bar: 0 ps Pointer: 195.65 ns Interval: 195.65 ns Start: End:

Figure 5. Part of the functional simulation.


Simulation Waveforms

Master Time Bar: 0 ps Pointer: 315.91 ns Interval: 315.91 ns Start: 317.7 ns End: 318.

Figure 6. Part of the timing simulation.

References

[1] Almaini, A. E. A. A semicustom IC for generation optimal generalized reed-Muller expansions. *Microelectronics Journal*, 28 (19970, 129-142.

[2] Drechsler, R., Theobald, M., and Becker, B. Fast OFDD-based minimization of fixed polarity Reed-Muller expansion. *IEEE Trans. Comp.*, 45 (1996), 1294-1299.

[3] Falkowski, B. J. and Chang, C. H. An exact minimization of fixed polarity Reed-Muller expansion. *Int. J. Electron.*, 79, 3 (1995), 389-409.

[4] Falkowski, B. J. and Chang, C. H. Hadamard-Walsh Spectral Characterization of Reed-Muller expansions. *Comp. Electr. Eng.*, 25 (1999), 111-134.

[5] Falkowski, B. J. and Chang, C. H. Minimization of k-variable mixed-polarity Reed-Muller expansions. *VLSI Design*, 11, 4 (2000), 311-320.

[6] Green, D. H. Reed-Muller canonical forms with mixed polarity and their manipulations. *IEE Proc., Part E*, 137 (1990), 103-113.

[7] Lui, P. K. and Muzio, J. C. Boolean matrix transforms for the minimization of modulo-2 canonical expressions. *IEEE Trans. Comp.*, 41 (1992), 342-347.

[8] Porwik, P. Efficient calculation of the Reed-Muller forms by means of the Walsh transform. *Int. J. Appl. Math. Comput. Sci.*, 12, 4 (2002), 571-579.

[9] Sasao, T. and Besslich, Ph. W. On the complexity of mod-2 sum PLAs. *IEEE Trans. Comp.*, 29 (1990), 262-266.

[10] Sasao, T. *Logic Synthesis and Optimization*. Kluwer Academic Publisher, 1993.

[11] Sasao, T. *Switching Theory for Logic Synthesis*. Kluwer Academic Publisher, 1999.

[12] Saul, J. Logic synthesis for arithmetic circuits using the Reed-Muller representation. In *Proc. Euro. Conf. Design Automation*, 1992, 109-113.

[13] Tsai, C-C. and Marek-Sadowska, M. Boolean functions classification via fixed polarity Reed-Muller forms. *IEEE Trans. Comp.*, 46, 2 (1997), 173-186.

[14] Tsai, C-C. and Marek-Sadowska, M. Boolean matching using generalized Reed-Muller forms. In *Proc. 31st Design Automation Conference*, (1994), 339-344.