A novel value-based multiplier architecture to multiply BCD numbers by powers of 10

Mohammad Samadi Gharajeh

Young Researchers and Elite Club,
Tabriz Branch,
Islamic Azad University,
Tabriz, Iran
Email: m.samadi@iaut.ac.ir
Email: mhm.samadi@gmail.com

Abstract: Multiplying the BCD numbers by powers of 10 is one of the essential operations in complex circuits. This paper proposes a novel multiplier to multiply BCD numbers by powers of 10. It uses a value-based architecture instead of the static architecture that is used in general multipliers. The proposed multiplier is composed of multiple embedded sub-multipliers so that each one multiplies one of the BCD numbers by the powers of 10. The multiplication results of each sub-multiplier are calculated by some mathematical equations based on input binary bits. Furthermore, the final output of the proposed multiplier are produced based the multiplication results of the sub-multipliers with the aid of a proposed selection unit. Comparison results demonstrate that the proposed multiplier surpasses some of the existing general multipliers in term of the number of logic gates. The proposed architecture is programmed by the VHDL codes and is simulated using the active-HDL simulation environment.

Keywords: electronic circuit; multiplication; multiplier; BCD number; power of 10; value-based architecture; computer architecture; computer system; arithmetic; circuit complexity; VHDL; active-HDL; simulation.

Reference to this paper should be made as follows: Gharajeh, M.S. (2022) 'A novel value-based multiplier architecture to multiply BCD numbers by powers of 10', *Int. J. Computer Aided Engineering and Technology*, Vol. 16, No. 3, pp.306–327.

Biographical notes: Mohammad Samadi Gharajeh received his ASc in Computer Software in 2005, BSc in Engineering of Computer Software Technology in 2009, and MSc in Computer Engineering – Computer Systems Architecture in 2013. His research interests include artificial intelligence software, robotics, internet of things, cloud computing, wireless sensor networks, and computer architecture. He was a Technical Program Committee member and a reviewer in some of the international conference proceedings. Besides, he is an editorial board member and a reviewer in some of the international scientific journals, a lecturer of university, an IEEE member, and an IAENG member.

1 Introduction

Multiplier (Kayal et al., 2014; Zhang et al., 2014) is one of the most popular circuits in digital electronics to multiply two binary numbers. Various types of the multipliers can be applied by digital applications including numerical systems, analogue devices, signal processing, and more. Whereas the multiplication process in typical processors has a long latency with a time between two and eight cycles (Law et al., 1999), it is essential that fast multipliers are considered in VLSI design of high-speed microprocessors. Number of logic gates, operation cycle, feature sizes, and die area are some of the main factors to evaluate the efficiency of the multipliers.

Multiplying the BCD numbers by powers of 10 is one of the necessary operations in electronic circuits. It can be used in various electronic architectures (e.g., decimal to binary converter). The existing general multipliers, in the most cases, have static structures to perform multiplication operations. They cause the number of logic gates to be considerably increased and, thereby, are not appropriate for some of the special electronic circuits. In this paper, a novel value-based multiplier is proposed to multiply BCD numbers by powers of 10. It works based on quantity amounts of the input binary bits instead of considering a general and complex structure. This multiplier leads the number of logic gates to be decreased and, therefore, power consumption of the digital circuits to be reduced, costs of the logic circuits to be alleviated, and delay time of the multiplication process to be decreased. Note that it can be applied in some of the existing multipliers to improve their complex operations.

The rest of this paper is organised as follows. Some of the existing multipliers are described in Section 2. Materials and circuits of the proposed multiplier are represented in Section 3. The proposed architecture is simulated by the VHDL codes in the active-HDL simulation environment as explained in Section 4. It is compared to some of the existing multipliers in term of the number of logic gates in Section 5. The multiplier structure is analysed in Section 6 to represent how it can be extended by other circuits. Finally, the paper is concluded in Section 7.

2 Background

A high performance 8 × 8 multiplier using Vedas is presented in Rajput et al. (2013). The authors have designed the multiplier by three circuits including CMOS technology, PTL logic, and MTCMOS technique. They have, also, summarised that a multiplier based on Vedic mathematics causes the delay time to be decreased and the power consumption to be reduced. The circuit of an improved 8-bit multiplier for an experimental RISC CPU is presented in Joshi et al. (2011). It is designed at the gate level based on a customised chip approach and supports the multiplication process of decimal numbers using an 8-bit integer unit and 16-bit floating point unit. Furthermore, some of the efficient algorithms are used in the suggested circuit to design an efficient multiplier. A power-efficient configurable Booth multiplier (CBM) is presented in Kuang and Wang (2010) to perform the single 8-bit, single 16-bit, and twin parallel 8-bit multiplication operations. It provides a flexible arithmetic capacity and a trade-off between output precision and power consumption. These goals are obtained by developing the various components including correcting-vector generator, a sign-bit generator, and a modified error compensation circuit.

Array multiplier (Lu, 2004) uses a simple architecture composed of the addition and shifting operations. It includes multiple partial multipliers generated by multiplying the multiplicand by the multiplier. The bit orders are used to shift the partial multipliers and the addition process is performed after the shifting stage. Wallace multiplier (Wallace, 1964) uses an efficient parallel multiplication algorithm. It has only a reduction stage delay of the order $O(\log n)$ so that the propagation delay of each layer is equal to $O(1)$. The partial products are performed in $O(1)$ and the final addition is performed in $O(\log n)$. Dadda multiplier (Dadda, 1965) operates similar to Wallace multiplier, but it contains a small number of logic gates. Meanwhile, it is slightly faster than Wallace multiplier in aspects of speed and cost. Reduced-area multiplier (Bickerstaff et al., 1993) operates like Wallace and Dadda multipliers, but it includes a maximum number of (3, 2) counter. This causes the number of binary bits to be minimised in the next reduction stage. Radix-4 Booth encoding multiplier (Shah et al., 2000) decreases the number of partial products for multiplying the multiplicand by the multiplier. However, the adding operations of the partial products cause the delay of the multiplication process to be increased. The delay time of the final product will be high due to increasing the number of binary bits in the multiplicand or the multiplier.

A scalable and hybrid 32-bit multiplier is presented in Kolla et al. (2003) composed of several linear array multipliers. It contains the performance capacity of three multipliers with the support of the higher-order multiplication processes. The performance results are obtained by two common design metrics including area and timing. A low-power 32-bit multiplier is presented in Jang et al. (2005) that uses pipelined block-wise shutdown. The static and dynamic powers are reduced by turning the supply voltage off through the idle stage. Meanwhile, the multiplier shutdowns and wakes up along with the pipeline stage. Performance of a 32-bit array multiplier with a carry save adder (CSA) and a carry-look-ahead adder (CLA) is analysed in Singh et al. (2009). CSA is used to perform the partial product lines as well as CLA is used to sum the partial product terms. The multiplier with CSA is better than that with CLA in terms of speed, area, and power consumption.

The modified booth algorithm and Wallace tree structure are used in Yao et al. (2011) to design a pipelined 32-bit multiplier. The propagation delay of the carry bits is reduced by CSA. The main goal of this multiplier is to reduce the resource consumption and the power consumption. Robertson's multiplier (Al Mijalli and Rais, 2012) performs the multiplication process of the operands with the aid of their sign bits. The final product is calculated in a series of the addition and shift stages when both operands are positive. The multiplier uses other processes when both operands are negative or one of them is negative. The multiplier presented in Dhanabal et al. (2013) uses CLA to minimise the calculation delay. It does not dissipate the power consumption to perform the multiplication processes.

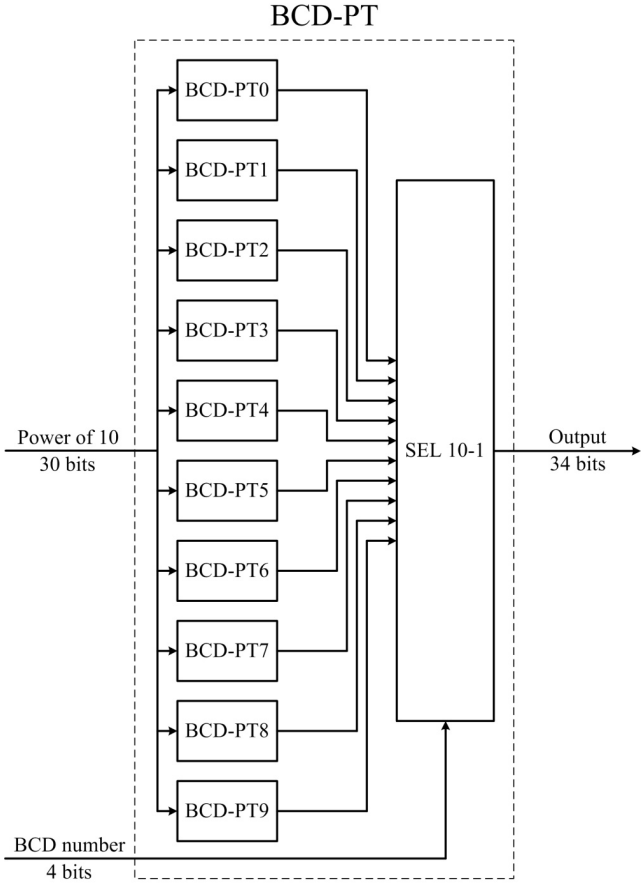
3 The proposed value-based multiplier architecture

Binary-coded decimal (BCD) is the decimal numbers 0 through 9 converted to four-digit binary bits. It is also known as packed decimal as represented in Table 1. In this paper, a novel value-based multiplier, called BCD-PT, is proposed to multiply BCD numbers by powers of 10. This multiplier is composed of the ten embedded sub-multipliers as shown in Figure 1.

Table 1 BCD numbers of the decimal numbers 0 through 9

Decimal number	*BCD number*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Figure 1 A schematic of the proposed multiplier architecture



Each sub-multiplier multiplies one of the BCD numbers by powers of 10 based on the input binary bits. Furthermore, a selector unit, called SEL 10-1, is proposed to transfer

multiplication results of one of the sub-multipliers to the output based on the input BCD number. All of the sub-multipliers contain different electronic circuits compared to each other in order to perform the required multiplication operations. The proposed architecture is composed of two inputs including power of 10 and BCD number. Power of 10 consists of thirty binary bits denoted by $P = \{p_{29}, p_{28}, \ldots, p_0\}$ as well as BCD number consists of four binary bits denoted by $B = \{b_3, b_2, b_1, b_0\}$. The P input includes the binary bits of 10^1 through 10^9. The multiplication results of the P and B inputs calculated by each sub-multiplier includes thirty-four binary bits denoted by $M = \{m_{33}, m_{32}, \ldots, m_0\}$. Moreover, the multiplication results of the proposed multiplier consists of thirty-four binary bits denoted by $O = \{o_{33}, o_{32}, \ldots, o_0\}$. The input bits of all the sub-multipliers are supplied by the P input bits. The internal circuits of the sub-multipliers and the selector unit are described as follows.

3.1 BCD-PT0

This sub-multiplier multiplies the decimal number 0 by the nine powers of 10. It gets BCD number by the series of $B = \{b_3, b_2, b_1, b_0\}$ and power of 10 by the series of $P = \{p_{29}, p_{28}, \ldots, p_0\}$ to calculate their multiplication results resulted by the series of $M = \{m_{33}, m_{32}, \ldots, m_0\}$. Table 2 represents the results of multiplying the decimal number 0 by the powers of 10 based on possible input binary bits. Because all of the multiplication results calculated by this sub-multiplier are equal to 0, the output bits of BCD-PT0 are determined as:

$$\forall i \in A, m_i = 0; \text{ where } A = \{0, 1, \ldots, 33\} \tag{1}$$

Table 2 Truth table for multiplying the decimal number 0 by the powers of 10

Symbol	*Inputs*		*Multiplication result*
	BCD number	*Power of 10*	
0×10^1	0000	00000000000000000000000001010	00000000000000000000000000000000
0×10^2	0000	00000000000000000000001100100	00000000000000000000000000000000
0×10^3	0000	00000000000000000001111101000	00000000000000000000000000000000
0×10^4	0000	00000000000000010011100010000	00000000000000000000000000000000
0×10^5	0000	00000000000011000011010100000	00000000000000000000000000000000
0×10^6	0000	00000000011110100001001000000	00000000000000000000000000000000
0×10^7	0000	00000010011000100101101000000	00000000000000000000000000000000
0×10^8	0000	00010111110101111000010000000	00000000000000000000000000000000
0×10^9	0000	11101110011010110010100000000	00000000000000000000000000000000

The number of basic logic gates including the AND, OR, NOT gates are one of the main parameters to evaluate the performance of the proposed multiplier. The number of logic gates applied by BCD-PT0 is specified as

$$N_{AND} = 0; N_{OR} = 0; N_{NOT} = 0; N_{Total} = N_{AND} + N_{OR} + N_{NOT} = 0 \tag{2}$$

where N_{AND} is the number of the AND gates, N_{OR} is the number of the OR gates, N_{NOT} is the number of the NOT gates, and N_{Total} is the number of all the logic gates in the BCD-PT0 sub-multiplier.

3.2 BCD-PT1

This sub-multiplier multiplies the decimal number 1 by the nine powers of 10. BCD number is entered in BCD-PT1 through the series of $B = \{b_3, b_2, b_1, b_0\}$ as well as power of 10 is entered through the series of $P = \{p_{29}, p_{28}, \ldots, p_0\}$. The results of the multiplication process are extracted from the sub-multiplier through the series of $M = \{m_{33}, m_{32}, \ldots, m_0\}$. As represented in Table 3, the multiplication results of the BCD-PT1 sub-multiplier are equal to the P input bits. Whereas the number of output bits is more than the number of power bits, the most significant four bits of the output are valued by the bit number 0 and the remainder bits of the output are equal to the P input bits. Therefore, the binary bits of the output are specified as:

$$\forall i \in A, m_i = p_i; \text{ where } A = \{0, 1, \ldots, 29\} \tag{3}$$

$$\forall j \in B, m_j = 0; \text{ where } B = \{30, 31, 32, 33\} \tag{4}$$

The number of logic gates used in BCD-PT1 is determined as:

$$N_{AND} = 0; N_{OR} = 0; N_{NOT} = 0; N_{Total} = 0 \tag{5}$$

where N_{AND} is the number of the AND gates, N_{OR} is the number of the OR gates, N_{NOT} is the number of the NOT gates, and N_{Total} is the number of all the logic gates in the BCD-PT1 sub-multiplier.

Table 3 Truth table for multiplying the decimal number 1 by the powers of 10

Symbol	*Inputs*		*Multiplication result*
	BCD number	*Power of 10*	
1×10^1	0001	00000000000000000000000001010	00000000000000000000000000001010
1×10^2	0001	00000000000000000000001100100	00000000000000000000000001100100
1×10^3	0001	00000000000000000001111101000	00000000000000000000001111101000
1×10^4	0001	00000000000000010011100010000	00000000000000000010011100010000
1×10^5	0001	00000000000011000011010100000	00000000000000011000011010100000
1×10^6	0001	00000000011110100001001000000	00000000000011110100001001000000
1×10^7	0001	00000100110001001011010000000	00000000010011000100101101000000
1×10^8	0001	00010111110101111000100000000	00000010111110101111000100000000
1×10^9	0001	11101110011010110010100000000	00001110111001101011001010000000

3.3 BCD-PT2

This sub-multiplier multiplies the decimal number 2 by the nine powers of 10. It gets BCD number by the series of $B = \{b_3, b_2, b_1, b_0\}$ and gets power of 10 by the series of $P = \{p_{29}, p_{28}, \ldots, p_0\}$. The multiplication results are emerged on the output by the series of $M = \{m_{33}, m_{32}, \ldots, m_0\}$. Table 4 represents the results of multiplying the decimal

number 2 by the powers of 10. As explained in this table, the multiplication result of each entry is equal to a logical left shift of one bit position of the P input bits. The reason is that the multiplication results are double the powers of 10. Note that the most significant three bits of the output are valued by the bit number 0. Hence, the binary bits of the output are calculated as:

$$\forall i \in A, m_i = p_{i-1}; \text{ where } A = \{1, 2, \ldots, 30\} \tag{6}$$

$$\forall j \in B, m_j = 0; \text{ where } B = \{0, 31, 32, 33\} \tag{7}$$

The number of logic gates applied by BCD-PT2 is determined as

$$N_{AND} = 0; N_{OR} = 0; N_{NOT} = 0; N_{Total} = 0 \tag{8}$$

where N_{AND} is the number of the AND gates, N_{OR} is the number of the OR gates, N_{NOT} is the number of the NOT gates, and N_{Total} is the number of all the logic gates in the BCD-PT2 sub-multiplier.

Table 4 Truth table for multiplying the decimal number 2 by the powers of 10

Symbol	*Inputs*		*Multiplication result*
	BCD number	*Power of 10*	
2×10^1	0010	00000000000000000000000001010	00000000000000000000000000010100
2×10^2	0010	00000000000000000000001100100	00000000000000000000000011001000
2×10^3	0010	00000000000000000001111101000	00000000000000000000011111010000
2×10^4	0010	00000000000000010011100010000	00000000000000000100111000100000
2×10^5	0010	00000000000011000011010100000	00000000000000001100011010100000
2×10^6	0010	00000000011110100001001000000	00000000000011110100001001000000
2×10^7	0010	00000100110001001011010000000	00000000100110001001011010000000
2×10^8	0010	00010111110101111000010000000	00000101111101011110000100000000
2×10^9	0010	11101110011010110010100000000	00011101110011010110010100000000

3.4 BCD-PT3

This sub-multiplier multiplies the decimal number 3 by the nine powers of 10. It obtains BCD number through the series of $B = \{b_3, b_2, b_1, b_0\}$ and power of 10 through the series of $P = \{p_{29}, p_{28}, \ldots, p_0\}$. Moreover, it transfers the multiplication results to the output through the series of $M = \{m_{33}, m_{32}, \ldots, m_0\}$. The output results for multiplying the decimal number 3 by the powers of 10 are represented in Table 5. As represented in this table, the multiplication results do not follow a regular manner. In fact, each binary bit in the output must be, independently, specified based on the P input bits.

Table 5 Truth table for multiplying the decimal number 3 by the powers of 10

Symbol	*Inputs*		*Multiplication result*
	BCD number	*Power of 10*	
3×10^1	0011	00000000000000000000000001010	00000000000000000000000000011110
3×10^2	0011	00000000000000000000000001100100	00000000000000000000000000100101100
3×10^3	0011	00000000000000000000001111101000	00000000000000000000000101110111000
3×10^4	0011	00000000000000000010011100010000	00000000000000000000111010100110000
3×10^5	0011	00000000000000011000011010100000	00000000000000000100100100111111000000
3×10^6	0011	00000000001111010000100100000000	00000000000001011011100011011000000
3×10^7	0011	00000010011000100101101000000000	00000000011100100111000011100000000
3×10^8	0011	00010111110101111000010000000000	00000100011110000110100011000000000
3×10^9	0011	11101110011010110010100000000000	00101100101101000001011110000000000

The binary bits of the results are determined based on the input binary bits as follows: some of the output bits are valued by the homologous input bits as:

$$\forall i \in A,\ m_i = p_i;\ \text{where } A = \{0, 1, 13, 23, 25, 29\} \tag{9}$$

Some of the output bits are filled by the bit number 0 as:

$$\forall i \in A,\ m_i = 0;\ \text{where } A = \{17, 26, 27, 30, 32, 33\} \tag{10}$$

Some of the output bits are determined by one of the input bits as:

$$\forall i \in A,\ m_i = p_{11};\ \text{where } A = \{20, 31\} \tag{11}$$

$$m_{21} = p_{18};\ m_{22} = p_{20} = m_{28} = p_{24} \tag{12}$$

Some of the output bits are determined based on the OR operation between various input bits as:

$$\begin{aligned} &m_2 = p_1 + p_2;\ m_3 = p_2 + p_3;\ m_4 = p_3 + p_4;\ m_5 = p_4 + p_5;\ m_{10} = p_4 + p_{17};\\ &m_{14} = p_4 + p_{19};\ m_{15} = p_{12} + p_{16};\ m_{16} = p_{12} + p_{18};\ m_{24} = p_{12} + p_{22} \end{aligned} \tag{13}$$

$$m_8 = p_2 + p_7 + p_8;\ m_9 = p_7 + p_{14} + p_{20} \tag{14}$$

One of the output bits are calculated based on the composition of the OR, AND operations between some of the input bits as:

$$m_{12} = p_4 + (p_5 p_{10}) + p_{11} \tag{15}$$

Some of the output bits are filled by the composition of the NOT, AND operations between some of the input binary bits as:

$$\forall i \in A,\ m_i = \overline{p_8} p_{16};\ \text{where } A = \{6, 18\} \tag{16}$$

$$m_{19} = \overline{p_{11}} p_{19} \tag{17}$$

Finally, one of the output bits is determined based on the composition of the NOT, AND, OR operations between some of the input bits as

$$m_7 = \left(\overline{p_5} p_6\right) + p_7; m_{11} = \left(\overline{p_1} p_3\right) + p_{11} \tag{18}$$

The XOR gate is composed of two AND gates, one OR gate, and two NOT gates. Hence, the number of all the logic gates used in BCD-PT3 is calculated as

$$N_{AND} = 6; N_{OR} = 17; N_{NOT} = 5; N_{Total} = 28 \tag{19}$$

where N_{AND} is the number of the AND gates, N_{OR} is the number of the OR gates, N_{NOT} is the number of the NOT gates, and N_{Total} is the number of all the logic gates in the BCD-PT3 sub-multiplier.

3.5 BCD-PT4

This sub-multiplier multiplies the decimal number 4 by the nine powers of 10. It calculates the multiplication operations between BCD number, obtained through the series of $B = \{b_3, b_2, b_1, b_0\}$, and power of 10, obtained through the series of $P = \{p_{29}, p_{28}, \ldots, p_0\}$. Moreover, it transfers the operation results to the output through the series of $M = \{m_{33}, m_{32}, \ldots, m_0\}$. As represented in Table 6, the multiplication results of the BCD-PT4 sub-multiplier are emerged on the output by the 34 bits. Whereas the results are four times the powers of 10, the output bits of each entry are equal to a logical left shift of two bits position of the P input bits. Therefore, the output bits of BCD-PT4 are determined as

$$\forall i \in A, m_i = p_{i-2}; \text{ where } A = \{2, 3, \ldots, 31\} \tag{20}$$

$$\forall j \in B, m_j = 0; \text{ where } B = \{0, 1, 32, 33\} \tag{21}$$

Table 6 Truth table for multiplying the decimal number 4 by the powers of 10

Symbol	*Inputs*		*Multiplication result*
	BCD number	*Power of 10*	
4×10^1	0100	00000000000000000000000001010	00000000000000000000000000101000
4×10^2	0100	00000000000000000000001100100	00000000000000000000000110010000
4×10^3	0100	00000000000000000001111101000	00000000000000000000111110100000
4×10^4	0100	00000000000000010011100010000	00000000000000001001110001000000
4×10^5	0100	00000000000011000011010100000	00000000000000110000110101000000
4×10^6	0100	00000000001111010000100100000	00000000000011110100001001000000000
4×10^7	0100	00000010011000100101101000000	00000000100110001001011010000000000
4×10^8	0100	00010111110101111000010000000	00000101111101011110000100000000000
4×10^9	0100	11101110011010110010100000000	00111011100110101100101000000000000

The number of logic gates utilised by BCD-PT4 is determined as

$$N_{AND} = 0; N_{OR} = 0; N_{NOT} = 0; N_{Total} = 0 \tag{22}$$

where N_{AND} is the number of the AND gates, N_{OR} is the number of the OR gates, N_{NOT} is the number of the NOT gates, and N_{Total} is the number of all the logic gates in the BCD-PT4 sub-multiplier.

3.6 BCD-PT5

This sub-multiplier multiplies the decimal number 5 by the nine powers of 10. It receives BCD number by the series of $B = \{b_3, b_2, b_1, b_0\}$ and power of 10 by the series of $P = \{p_{29}, p_{28}, \ldots, p_0\}$ to calculate the multiplication results extracted to the output by the series of $M = \{m_{33}, m_{32}, \ldots, m_0\}$. Table 7 represents the multiplication results of BCD-PT5 that is explained by the 34 binary bits. It is clear that these results have not the uniform form so that each one must be, separately, calculated by a relation between BCD number and power of 10.

Table 7 Truth table for multiplying the decimal number 5 by the powers of 10

Symbol	*Inputs*		*Multiplication result*
	BCD number	*Power of 10*	
5×10^1	0101	00000000000000000000000001010	00000000000000000000000000110010
5×10^2	0101	00000000000000000000001100100	00000000000000000000000111110100
5×10^3	0101	00000000000000000001111101000	00000000000000000000010011100010000
5×10^4	0101	00000000000000010011100010000	00000000000000000011000011010100000
5×10^5	0101	00000000000011000011010100000	00000000000000011110100001001000000
5×10^6	0101	00000000011110100001001000000	00000000000100110001001011010000000
5×10^7	0101	00000010011000100101101000000	00000000101111101011110000100000000
5×10^8	0101	00010111110101111000010000000	00000111011100110101100101000000000
5×10^9	0101	11101110011010110010100000000	01001010100000010111110010000000000

Some of the output bits are determined according to the homologous input bits as:

$$\forall i \in A, m_i = p_i; \text{ where } A = \{0, 1, 2, 26, 29\} \tag{23}$$

Some of the output bits are filled by the bit number 0:

$$\forall i \in A, m_i = 0; \text{ where } A = \{30, 31, 33\} \tag{24}$$

Some of the outputs are specified by one of the input binary bits as:

$$\forall i \in A, m_i = p_{21}; \text{ where } A = \{10, 24, 28\} \tag{25}$$

$$\forall j = B, m_j = p_{12}; \text{ where } B = \{20, 21\} \tag{26}$$

$$m_{13} = p_{15}; m_{27} = p_{24}; m_{32} = p_{11} \tag{27}$$

Some of the output bits are determined by the OR operation between multiple input bits as

$$\forall i \in A, m_i = p_{12} + p_{18}; \text{ where } A = \{19, 22\} \tag{28}$$

$$m_{15} = p_{10} + p_{25}; m_{18} = p_{11} + p_{16}; m_{23} = p_{12} + p_{21}; m_{25} = p_{12} + p_{25} \tag{29}$$

$$m_4 = p_1 + p_2 + p_4;\ m_8 = p_6 + p_8 + p_{16};\ m_{14} = p_{12} + p_{13} + p_{14} \tag{30}$$

One of the output bits is determined by the AND operation between two input bits as:

$$m_3 = p_3 p_6 \tag{31}$$

Some of the output bits are specified by the composition of the AND, NOT operations between some of the input bits as:

$$m_{11} = p_{18}\overline{p_{21}};\ m_{17} = p_{15}\overline{p_{24}} \tag{32}$$

Some of the output bits are valued by the composition of the OR, XOR operations between some of the input bits as:

$$m_5 = (p_1 + p_2 + p_{15}) \oplus p_{20};\ m_6 (p_2 + p_4 + p_{18}) \oplus p_{26} \tag{33}$$

Eventually, the remainder output bits are calculated based on the composition of the NOT, AND, OR operations between multiple input binary bits as:

$$\begin{aligned} &m_7 = (\overline{p_1} p_3) + p_2 + p_{12};\ m_9 = (p_8 + p_{17})\overline{p_{21}};\ m_{12} = (\overline{p_1} p_3) + p_{11} + p_{12}; \\ &m_{16} = (p_{15}\overline{p_{20}}) + p_{24} \end{aligned} \tag{34}$$

The number of logic gates utilised by BCD-PT5 is determined as

$$N_{AND} = 11;\ N_{OR} = 23;\ N_{NOT} = 10;\ N_{Total} = 44 \tag{35}$$

where N_{AND} is the number of the AND gates, N_{OR} is the number of the OR gates, N_{NOT} is the number of the NOT gates, and N_{Total} is the number of all the logic gates in the BCD-PT5 sub-multiplier.

3.7 BCD-PT6

This sub-multiplier multiplies the decimal number 6 by the nine powers of 10. It gets BCD number through the series of $B = \{b_3, b_2, b_1, b_0\}$ and power of 10 through the series of $P = \{p_{29}, p_{28}, \ldots, p_0\}$. The multiplication results are emerged in the output through the series of $M = \{m_{33}, m_{32}, \ldots, m_0\}$. The multiplication operations of BCD-PT6 are performed based on the quantity amounts represented in Table 8. As explained in this table, the output bits cannot be determined based on a uniform procedure. Therefore, they must be, independently, determined based on the powers of 10.

Some of the output bits are valued by the bit number 0 as:

$$\forall i \in A,\ m_i = 0;\ \text{where } A = \{0, 1, 18, 27, 28, 31, 33\} \tag{36}$$

Some of the output bits are determined by one of the input bits as:

$$\forall i \in A,\ m_i = p_{11};\ \text{where } A = \{21, 26, 30, 32\} \tag{37}$$

$$\forall j \in B,\ m_j = p_{20};\ \text{where } B = \{23, 24\} \tag{38}$$

$$m_2 = p_1;\ m_{14} = p_{13};\ m_{22} = p_{18};\ m_{29} = p_{24} \tag{39}$$

Some of the output bits are calculated based on the OR operation between multiple input bits as:

$$m_3 = p_1 + p_2; m_5 = p_3 + p_4; m_6 = p_4 + p_5; m_{10} = p_7 + p_{14}; m_{11} = p_4 + p_{17}; \\ m_{15} = p_4 + p_{19}; m_{16} = p_{12} + p_{16}; m_{17} = p_{12} + p_{18}; m_{25} = p_{12} + p_{21} \tag{40}$$

$$m_4 = p_1 + p_2 + p_3; m_9 = p_2 + p_7 + p_8 \tag{41}$$

Some of the output bits are specified based on the composition of the NOT, AND operations between some of the input binary bits as:

$$\forall i \in A, m_i = \overline{p_{13}} p_{16}; \text{ where } A = \{7, 19\} \tag{42}$$

Some of the output bits are determined by the composition of the OR, AND, NOT operations between several input bits as:

$$m_8 = (p_7 + p_{14})\overline{p_{24}}; m_{13} = p_4 + (p_5 p_{10}) + p_{11} \tag{43}$$

Some of the output bits are calculated by the composition of the AND, OR operations between some of the input binary bits as:

$$m_{12} = (p_3 p_5) + p_{11}; m_{13} = p_4 + (p_5 p_{10}) + p_{11} \tag{44}$$

The number of logic gates used in BCD-PT6 is specified as:

$$N_{AND} = 6; N_{OR} = 18; N_{NOT} = 4; N_{Total} = 28 \tag{45}$$

where N_{AND} is the number of the AND gates, N_{OR} is the number of the OR gates, N_{NOT} is the number of the NOT gates, and N_{Total} is the number of all the logic gates in the BCD-PT6 sub-multiplier.

Table 8 Truth table for multiplying the decimal number 6 by the powers of 10

Symbol	*Inputs*		*Multiplication result*
	BCD number	*Power of 10*	
6×10^1	0110	00000000000000000000000001010	00000000000000000000000000111100
6×10^2	0110	00000000000000000000001100100	00000000000000000000000001001011000
6×10^3	0110	00000000000000000001111101000	00000000000000000000010111011110000
6×10^4	0110	00000000000000010011100010000	00000000000000000011101010011100000
6×10^5	0110	00000000000011000011010100000	00000000000000100100100111111000000
6×10^6	0110	00000000011111010000100100000	00000000000101101110001101100000000
6×10^7	0110	00000100110001001011010000000	00000000111001001110000111000000000
6×10^8	0110	00010111110101111000010000000	00001000111100001101000110000000000
6×10^9	0110	11101110011010110010100000000	01011001011010000010111100000000000

3.8 BCD-PT7

This sub-multiplier multiplies the decimal number 7 by the nine powers of 10. It receives BCD number by the series of $B = \{b_3, b_2, b_1, b_0\}$ and power of 10 by the series of

$P = \{p_{29}, p_{28}, \ldots, p_0\}$ to produce the multiplication results in the output by the series of $M = \{m_{33}, m_{32}, \ldots, m_0\}$. Table 9 represents the quantity amounts of the multiplication process performed by this sub-multiplier. Because the multiplication results calculated by BCD-PT7 are not uniform, each output binary bit must be, separately, specified based on the input binary bits.

Table 9 Truth table for multiplying the decimal number 7 by the powers of 10

Symbol	*Inputs*		*Multiplication result*
	BCD number	*Power of 10*	
7×10^1	0111	00000000000000000000000000001010	00000000000000000000000000001000110
7×10^2	0111	00000000000000000000000001100100	00000000000000000000000001010111100
7×10^3	0111	00000000000000000000001111101000	00000000000000000000011101101011000
7×10^4	0111	00000000000000000100111000010000	00000000000000000100010001011110000
7×10^5	0111	00000000000001100001101010000000	00000000000000101010101110011000000
7×10^6	0111	00000000001111010000100100000000	00000000000110101011001111111000000
7×10^7	0111	00000010011000100101101000000000	00000001000010110000011101100000000
7×10^8	0111	00010111110101111000010000000000	00001010011011100100100111000000000
7×10^9	0111	11101110011010110010100000000000	01101000010011101110000110000000000

Some of the output bits are set by the bit number 0 as:

$$\forall i \in A, m_i = 0; \text{ where } A = \{0, 25, 28, 30, 33\} \tag{46}$$

Some of the output bits are specified based on one of the input binary bits as:

$$m_1 = p_1 \tag{47}$$

$$\forall i \in A, m_i = p_{12}; \text{ where } A = \{18, 26\} \tag{48}$$

$$\forall j \in B, m_j = p_{24}; \text{ where } B = \{20, 24, 29\} \tag{49}$$

$$\forall k \in C, m_k = p_{21}; \text{ where } C = \{23, 27\} \tag{50}$$

$$\forall l \in D, m_l = p_{11}; \text{ where } D = \{31, 32\} \tag{51}$$

Some of the output bits are calculated by the OR operation between multiple input binary bits as:

$$m_2 = p_1 + p_2; m_9 = p_5 + p_{14}; m_{16} = p_4; m_{21} = p_{12} + p_{14} \tag{52}$$

$$m_8 = p_8 + p_{12} + p_{18} \tag{53}$$

$$\forall i \in A, m_i = p_{15} + p_{17}; \text{ where } A = \{10, 19\} \tag{54}$$

Some of the output bits are determined by the composition of the AND, NOT operations between some of the input bits as:

$$m_3 = p_5\,\overline{p_{10}};\ m_{13} = \overline{p_6}\,p_{16} \tag{55}$$

$$\forall i \in A,\ m_i = \overline{p_{11}}\,p_{17};\ \text{where } A = \{14, 22\} \tag{56}$$

Finally, some of the output bits are calculated by the composition of the NOT, AND, OR operations between input binary bits as:

$$\begin{aligned} &m_5 = (p_2 + p_{10})\,\overline{p_{12}};\ m_6 = (p_1 + p_9)\,\overline{p_{20}};\ m_7 = (p_2 + p_{19})\,\overline{p_{11}};\\ &m_{11} = (p_7 + p_{16})\,\overline{p_{21}} \end{aligned} \tag{57}$$

$$m_4 = (p_2 + p_3 + p_4)\,\overline{p_1};\ m_{12} = (p_3 + p_4 + p_{12})\,\overline{p_1} \tag{58}$$

$$\forall i \in A,\ m_i = \left(\overline{p_{13}}\,p_{16}\right) + p_{11};\ \text{where } A = \{15, 17\} \tag{59}$$

The number of logic gates applied by BCD-PT7 is determined as:

$$N_{AND} = 12;\ N_{OR} = 18;\ N_{NOT} = 12;\ N_{Total} = 42 \tag{60}$$

where N_{AND} is the number of the AND gates, N_{OR} is the number of the OR gates, N_{NOT} is the number of the NOT gates, and N_{Total} is the number of all the logic gates in the BCD-PT7 sub-multiplier.

3.9 BCD-PT8

This sub-multiplier multiplies the decimal number 8 by the nine powers of 10. It obtains BCD number by the series of $B = \{b_3, b_2, b_1, b_0\}$ and power of 10 by the series of $P = \{p_{29}, p_{28}, \ldots, p_0\}$. The multiplication results are produced by the series of $M = \{m_{33}, m_{32}, \ldots, m_0\}$. Table 10 represents the results of the multiplication operations between the decimal number 8 and the powers of 10. As explained in this table, the multiplication result of each entry is equal to a logical left shift of three bits position of the P input bits. The reason is that the multiplication results are eight times the powers of 10. Note that the most significant three bits of the results are valued by the bit number 0. Thus, the output binary bits are determined as:

$$\forall i \in A,\ m_i = p_{i-3};\ \text{where } A = \{3, 4, \ldots, 32\} \tag{61}$$

$$\forall j \in B,\ m_j = 0;\ \text{where } B = \{0, 1, 2, 33\} \tag{62}$$

The number of logic gates utilised by BCD-PT8 is specified as:

$$N_{AND} = 0;\ N_{OR} = 0;\ N_{NOT} = 0;\ N_{Total} = 0 \tag{63}$$

where N_{AND} is the number of the AND gates, N_{OR} is the number of the OR gates, N_{NOT} is the number of the NOT gates, and N_{Total} is the number of all the logic gates in the BCD-PT8 sub-multiplier.

Table 10 Truth table for multiplying the decimal number 8 by the powers of 10

Symbol	*Inputs*		*Multiplication result*
	BCD number	*Power of 10*	
8×10^1	1000	00000000000000000000000001010	00000000000000000000000001010000
8×10^2	1000	00000000000000000000001100100	00000000000000000000001100100000
8×10^3	1000	00000000000000000001111101000	00000000000000000001111101000000
8×10^4	1000	00000000000000010011100010000	00000000000000010011100010000000
8×10^5	1000	00000000000011000011010100000	00000000000011000011010100000000
8×10^6	1000	00000000011110100001001000000	00000000011110100001001000000000
8×10^7	1000	00000100110001001011010000000	00000100110001001011010000000000
8×10^8	1000	00010111110101111000010000000	00010111110101111000010000000000
8×10^9	1000	11101110011010110010100000000	01110111001101011001010000000000

3.10 BCD-PT9

This sub-multiplier multiplies the decimal number 9 by the nine powers of 10. BCD number is entered in the sub-multiplier through the series of $B = \{b_3, b_2, b_1, b_0\}$ and power of 10 is entered in the sub-multiplier by the series of $P = \{p_{29}, p_{28}, \ldots, p_0\}$ to produce the multiplication results in the output by the series of $M = \{m_{33}, m_{32}, \ldots, m_0\}$. The multiplication process between the decimal number 9 and the powers of 10 are resulted in Table 11. The multiplication results indicate that the output binary bits of BCD-PT9 do not follow the same form. Each binary bit, thereby, must be determined regarding the powers of 10.

Table 11 Truth table for multiplying the decimal number 9 by the powers of 10

Symbol	*Inputs*		*Multiplication result*
	BCD number	*Power of 10*	
9×10^1	1001	00000000000000000000000001010	00000000000000000000000001011010
9×10^2	1001	00000000000000000000000001100100	00000000000000000000000001110000100
9×10^3	1001	00000000000000000000001111101000	00000000000000000000010001100101000
9×10^4	1001	00000000000000000100111000100000	00000000000000000101011111110010000
9×10^5	1001	00000000000000110000110101000000	00000000000000110110111011101000000
9×10^6	1001	00000000001111010000100100000000	00000000001000100101010100010000000
9×10^7	1001	00000010011000100101101000000000	00000001010101110101001010100000000
9×10^8	1001	00010111110101111000010000000000	00001101011010010011101001000000000
9×10^9	1001	11101110011010110010100000000000	10000110000111000100011010000000000

Some of the output binary bits are equal to the bit number 0 as:

$$\forall i \in A, m_i = 0; \text{ where } A = \{0, 17, 25, 30, 31, 32\} \tag{64}$$

Some of the output bits are determined based on the homologous binary bits as:

$$\forall i \in A, m_i = p_i; \text{ where } A = \{1, 2, 3, 27\} \tag{65}$$

Some of the output bits are specified by one of the input binary bits as:

$$\forall i \in A, m_i = p_{24}; \text{ where } A = \{21, 28\} \tag{66}$$

$$m_{23} = p_{18}; m_{29} = p_{21}; m_{33} = p_{11} \tag{67}$$

Some of the output bits are valued by the OR operation between input binary bits as:

$$\forall i \in A, m_i = p_{11} + p_{12}; \text{ where } A = \{20, 22\} \tag{68}$$

$$\forall j \in B, m_j = p_{12} + p_{21}; \text{ where } B = \{24, 26\} \tag{69}$$

$$m_4 = p_1 + p_4; m_7 = p_2 + p_{10}; m_8 = p_5 + p_8; m_{11} = p_{10} + p_{15} \tag{70}$$

$$m_{14} = p_{12} + p_{13} + p_{18} \tag{71}$$

$$m_9 = p_4 + p_5 + p_{12} + p_{25} \tag{72}$$

Some of the output bits are calculated based on the composition of the AND, NOT operations between some of the input bits as:

$$m_5 = \overline{p_2} p_5; m_{15} = \overline{p_6} p_{16}; m_{16} = \overline{p_3} p_9 = m_{18} = p_{15} \overline{p_{25}} \tag{73}$$

Some of the output bits are determined by the composition of the AND, OR operations between input bits as:

$$m_{12} = p_4 + (p_5 p_{10}) + p_{17} \tag{74}$$

Ultimately, some of the output bits are specified based on the composition of the NOT, AND, OR operations between some of the input binary bits as:

$$m_6 = (p_1 + p_{17}) \overline{p_{25}}; m_{10} = (p_4 + p_{17}) \overline{p_{25}} \tag{75}$$

$$m_{13} = (\overline{p_2} p_5) + p_{21}; m_{19} = (\overline{p_8} p_{16}) + p_{12} \tag{76}$$

The number of logic gates used in BCD-PT9 is resulted as

$$N_{AND} = 9; N_{OR} = 19; N_{NOT} = 8; N_{Total} = 36 \tag{77}$$

where N_{AND} is the number of the AND gates, N_{OR} is the number of the OR gates, N_{NOT} is the number of the NOT gates, and N_{Total} is the number of all the logic gates in the BCD-PT9 sub-multiplier.

3.11 SEL 10-1

This unit operates like a selector to transform the multiplication results of the sub-multipliers to the output based on BCD number. It works similar to the multiplexer 16 to 1 (Midtgaard and Svensson, 1994), but it has only ten input bits. SEL 10-1 gets BCD number by the series of $B = \{b_3, b_2, b_1, b_0\}$ and the multiplication results by the series of $M = \{m_{33}, m_{32}, \ldots, m_0\}$ to produce the final multiplication results of the proposed multiplier by the series of $O = \{o_{33}, o_{32}, \ldots, o_0\}$. As represented in Table 12, each BCD number specifies the multiplication results of which sub-multiplier must to be transformed to the output. The internal architecture of the SEL 10-1 unit to produce one of the output binary bits is shown in Figure 2.

Table 12 The selection process of the sub-multipliers in the SEL 10-1 unit based on BCD number

BCD number	*Selected sub-multiplier*
0000	BCD-PT0
0001	BCD-PT1
0010	BCD-PT2
0011	BCD-PT3
0100	BCD-PT4
0101	BCD-PT5
0110	BCD-PT6
0111	BCD-PT7
1000	BCD-PT8
1001	BCD-PT9

Because the output binary bits of the proposed multiplier are produced based on the multiplication results of the sub-multipliers via the same procedure, the algebraic form of the SEL 10-1 unit is represented as:

$$\begin{aligned}\forall i \in A,\ o_i = &\left(\overline{b_3}\,\overline{b_2}\,\overline{b_1}\,\overline{b_0} m_i^{BCD\text{-}PT0}\right) + \left(\overline{b_3}\,\overline{b_2}\,\overline{b_1} b_0 m_i^{BCD\text{-}PT1}\right) + \left(\overline{b_2} b_1 \overline{b_0} m_i^{BCD\text{-}PT2}\right) \\ &+ \left(\overline{b_2} b_1 b_0 m_i^{BCD\text{-}PT3}\right) + \left(b_2 \overline{b_1}\,\overline{b_0} m_i^{BCD\text{-}PT4}\right) + \left(b_2 \overline{b_1} b_0 m_i^{BCD\text{-}PT5}\right) \\ &+ \left(b_2 b_1 \overline{b_0} m_i^{BCD\text{-}PT6}\right) + \left(b_2 b_1 b_0 m_i^{BCD\text{-}PT7}\right) + \left(b_3 \overline{b_2}\,\overline{b_1}\,\overline{b_0} m_i^{BCD\text{-}PT8}\right) \\ &+ \left(b_3 \overline{b_2}\,\overline{b_1} b_0 m_i^{BCD\text{-}PT0}\right);\ \text{where } A = \{0, 1, \ldots, 33\}\end{aligned} \tag{78}$$

where i indicates each output binary bit of the proposed multiplier. The number of logic gates used in SEL 10-1 for each output binary bit is determined as:

$$N_{AND} = 34;\ N_{OR} = 9;\ N_{NOT} = 19;\ N_{Total} = 52 \tag{79}$$

where N_{AND} is the number of the AND gates, N_{OR} is the number of the OR gates, N_{NOT} is the number of the NOT gates, and N_{Total} is the number of all the logic gates in the SEL 10-1 unit for one output bit. Therefore, the number of logic gates used in SEL 10-1 for all of the output binary bits is calculated as:

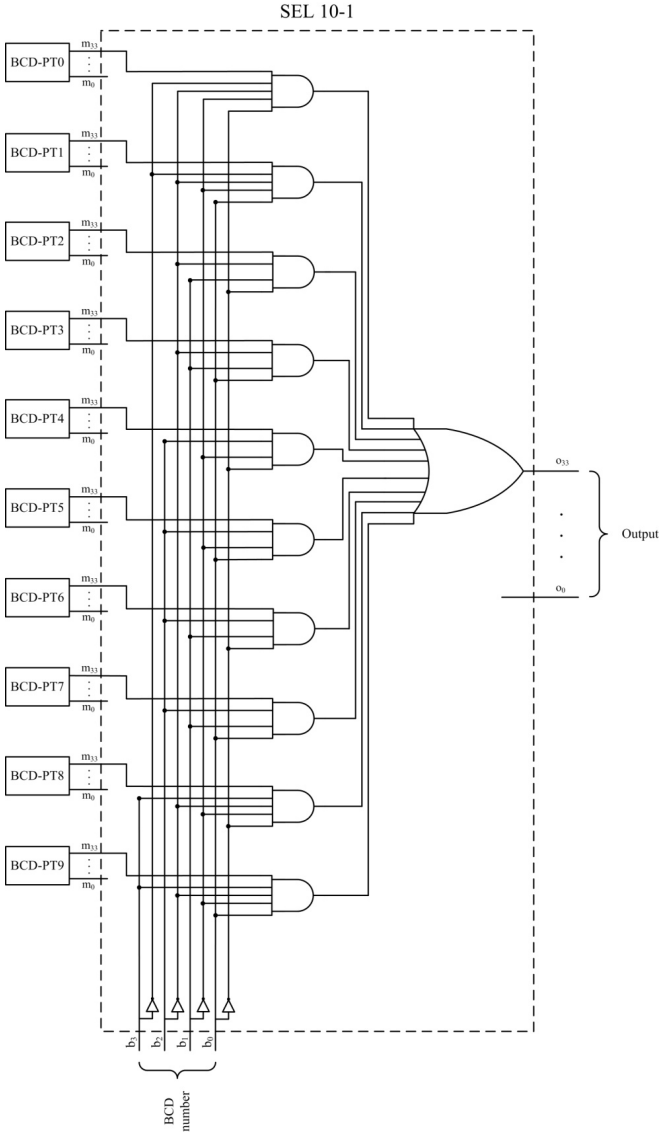
$$N_{AND} = 1{,}156;\ N_{OR} = 306;\ N_{NOT} = 646;\ N_{Total} = 2{,}108 \tag{80}$$

where N_{AND} is the number of the AND gates, N_{OR} is the number of the OR gates, N_{NOT} is the number of the NOT gates, and N_{Total} is the number of all the logic gates in the SEL 10-1 unit for all of the output bits. According to equations (2), (5), (8), (19), (22), (35), (45), (60), (63), (77), and (80), the number of basic logic gates used in the proposed multiplier is calculated as:

$$N_{AND} = 1{,}200;\ N_{OR} = 401;\ N_{NOT} = 685;\ N_{Total} = 2{,}286 \tag{81}$$

where N_{AND} is the number of the AND gates, N_{OR} is the number of the OR gates, N_{NOT} is the number of the NOT gates, and N_{Total} is the number of all the logic gates in the proposed multiplier.

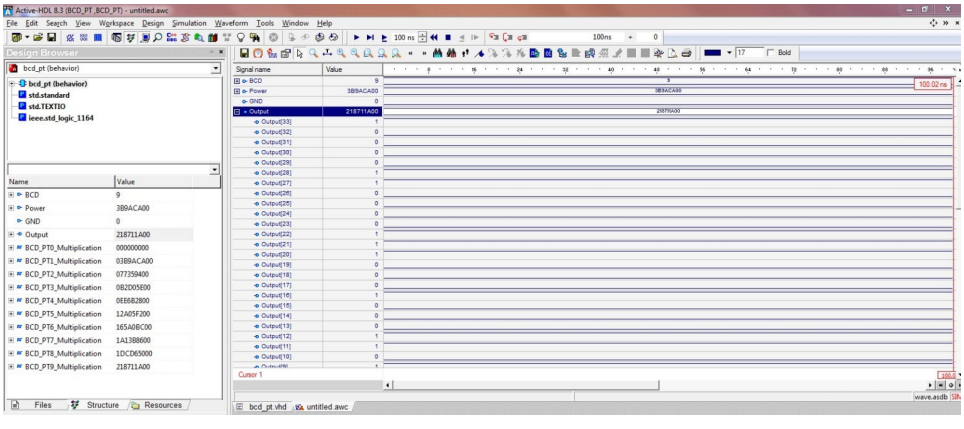
Figure 2 A schematic of the SEL 10-1 unit to determine one of the output bits



4 Simulation results

The proposed multiplier is simulated and analysed in the Active-HDL v8.3 SP1 by the VHDL (Shi et al., 1994) codes. Whereas it is composed of several internal units, each unit is designed as the unique entity to can be used into other entities. Schematics of the entities are generated by the Code2Graphics conversion tool. An extra input binary bit, called GND, is defined into the entities containing the bit number 0 due to some of the output bits are equal to the bit number 0. For example, the waveforms of the control signals designed by the active-HDL for multiplying the decimal number 9 by 10^9 are illustrated in Figure 3. BCD number is determined as '100', power of 10 is initialised by '11101110011010110010100000000', and the GND input is set by '0'. The multiplication results are calculated as '1000011000011100010001101000000000' by the multiplier, which represent the integer number 9×10^9.

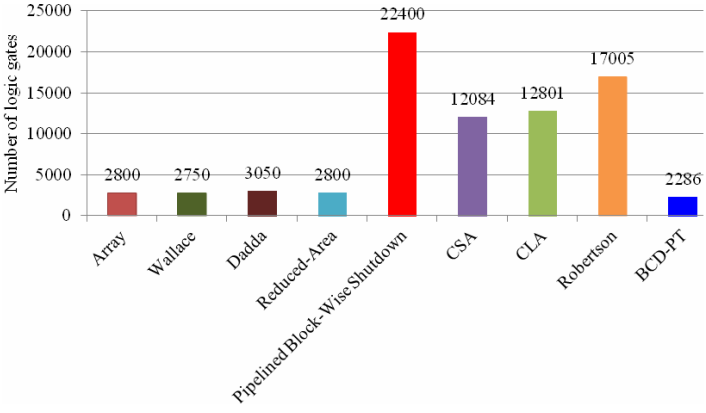
Figure 3 The waveforms of the control signals to multiply the decimal number 9 by 10^9 (see online version for colours)



5 Comparison results

The main goal of this paper is to design a novel architecture for multiplying the BCD number by power of 10 having a few number of logic gates. Until now, the general 32-bit multipliers are used to perform this operation. The proposed multiplier is compared to array (Swee and Hiung, 2012), Wallace (Swee and Hiung, 2012), Dadda (Swee and Hiung, 2012), reduced-area (Bickerstaff et al., 1993), pipelined block-wise shutdown (Jang et al., 2005), CSA (Singh et al., 2009), CLA (Singh et al., 2009; Dhanabal et al., 2013) and Robertson (Al Mijalli and Rais, 2012) multipliers in term of the number of logic gates as shown in Figure 4. Comparison results demonstrate that the number of logic gates in the BCD-PT multiplier is less than that in the other multipliers.

Figure 4 Performance of the proposed multiplier compared to some of the existing multipliers (see online version for colours)



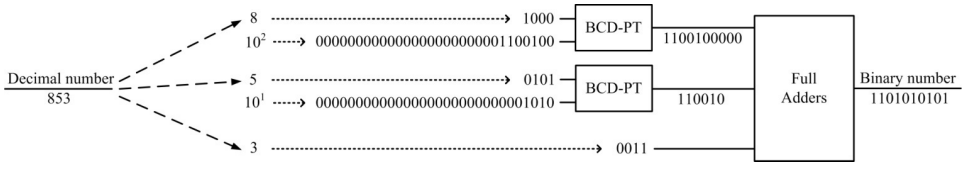
6 Discussion and extended works

Decimal to binary converter is one of the most popular integrated chips that can be designed by the proposed multiplier in order to facilitate the internal electronic circuits. Decimal numbers are very applicable in the real world, but they should be converted to binary number to can be used by microprocessors. For example, the decimal number 853 can be converted to binary number via the principle of the positional weighting as:

$$\begin{aligned} 853 &= 8\times 10^2 + 5\in 10^1 + 3 \\ &= 1100100000 + 110010 + 11 \\ &= 1101010101 \end{aligned}$$

The above number conversion can be performed by the proposed multiplier and two full adders as shown in Figure 5. Each digit of the decimal number 853 is converted to a BCD number. Afterward, it is multiplied by an appropriate power of 10 according to its position at the decimal number. Finally, the multiplications results and the least significant digit number are added with together by two full adders. The analysis results represent that the number of logic gates and the calculation process achieved by this procedure are less than those achieved by other procedures designed by existing general multipliers. Note that the proposed multiplier can be used in the internal circuits of the existing multipliers to reduce their complexity. This leads the financial costs of the circuits to be economised and the efficiency of the multipliers to be enhanced, considerably.

Figure 5 A schematic of the number conversion process from decimal system to binary system



7 Conclusions and future work

Multiplier is one of the most popular units in the electronic circuits. The existing general multipliers, in the most cases, operate based on the static architectures which are used to multiply all of the possible inputs. They cause the number of logic gates to be increased and the delay time of the operations to be enhanced, considerably. In this paper, a novel value-based multiplier, called BCD-PT, is proposed to multiply BCD number by powers of 10. This multiplier is composed of several sub-multipliers so that each one multiplies one of the BCD numbers by the powers of 10. The multiplication results of the multiplier are produced based on the inputs by a proposed selector unit. The BCD-PT multiplier is simulated in the Active-HDL simulator using the VHDL codes. Comparison results show that the number of logic gates used in the proposed multiplier is nearly 20% less than that used in the array multiplier, nearly 15% less than that used in the Wallace multiplier, nearly 25% less than that used in the Dadda multiplier, nearly 20% less than that used in the reduced-area multiplier, nearly 100% less than that used in the pipelined block-wise shutdown multiplier, nearly 80% less than that used in the CSA multiplier, nearly 85% less than that used in the CLA multiplier, and nearly 85% less than that used in the Robertson multiplier. In the future work, more powers of 10 will be considered by the proposed multiplier architecture. It can be used to multiply larger numbers and to convert larger decimal numbers to binary numbers.

References

Al Mijalli, M.H. and Rais, M.H. (2012) 'Hardware realization of Robertson's multiplier', *Middle-East Journal of Scientific Research*, Vol. 12, No. 1, pp.1–5.

Bickerstaff, K.A.C., Schulte, M. and Swartzlander Jr., E.E. (1993) 'Reduced area multipliers', *Proceedings of the IEEE International Conference on Application-Specific Array Processors*, 25–27 October, Venice, pp.478–489.

Dadda, L. (1965) 'Some schemes for parallel multipliers', *Alta Frequenza*, Vol. 34, No. 5, pp.349–356.

Dhanabal, R., Bharathi, V., Anand, N., Joseph, G., Oommen, S.S. and Sahoo, S.K. (2013) 'Comparison of existing multipliers and proposal of a new design for optimized performance', *International Journal of Engineering and Technology (IJET)*, Vol. 5, No. 2, pp.1704–1709.

Jang, Y-J., Shin, Y., Hong, M-C., Wee, J.K. and Lee, S. (2005) 'Low-power 32bit×32bit multiplier design with pipelined block-wise shutdown', *High Performance Computing – HiPC*, pp.398–406, Springer, Berlin, Heidelberg.

Joshi, A., Lam, S. and Chan, Y. (2011) 'Design of an improved multiplier unit for an experimental RISC CPU', *Electrical Power Systems and Computers*, Vol. 99, pp.323–330, Springer, Berlin, Heidelberg.

Kayal, D., Mostafa, P., Dandapat, A. and Sarkar, C.K. (2014) 'Design of high performance 8 bit multiplier using Vedic multiplication algorithm with McCMOS technique', *Journal of Signal Processing Systems*, Vol. 76, No. 1, pp.1–9.

Kolla, Y., Kim, Y-B. and Carter, J. (2003) 'A novel 32-bit scalable multiplier architecture', *Proceedings of the 13th ACM Great Lakes Symposium on VLSI*, New York, NY, USA, pp.241–244.

Kuang, S-R. and Wang, J-P. (2010) 'Design of power-efficient configurable booth multiplier', *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 57, No. 3, pp.568–580.

Law, C.F., Rofail, S.S. and Yeo, K.S. (1999) 'A low-power 16×16-b parallel multiplier utilizing pass-transistor logic', *IEEE Journal of Solid-State Circuits*, Vol. 34, No. 10, , pp.1395–1399.

Lu, M. (2004) 'Modular structure of large multiplier', *Arithmetic and Logic in Computer Systems*, John Wiley & Sons, Hoboken, New Jersey, USA.

Midtgaard, J. and Svensson, C. (1994) '5.8 Gb/s 16:1 multiplexer and 1:16 demultiplexer using 1.2 μm BiCMOS', *Proceedings of the IEEE International Symposium on Circuits and Systems*, 30 May–2 June, London, Vol. 4, pp.43–46.

Rajput, N., Sethi, M., Dobriyal, P., Sharma, K. and Sharma, G. (2013) 'A novel, high performance and power efficient implementation of 8×8 multiplier unit using MT-CMOS technique', *Proceedings of the IEEE 6th International Conference on Contemporary Computing (IC3)*, 8–10 August, Noida, pp.186–191.

Shah, S., Al-Khalili, A.J. and Khalili, D. (2000) 'Comparison of 32-bit multipliers for various performance measures', *Proceedings of the IEEE 12th International Conference on Microelectronics*, Tehran, pp.75–80.

Shi, R., Christen, E., Liebmann, P., Krolikoski, S. and Zhou, W. (1994) 'VHDL-A: analog extension to VHDL', *Proceedings of the 7th Annual IEEE International ASIC Conference and Exhibit*, 19–23 September, Rochester, NY, pp.160–165.

Singh, R.P.P., Kumar, P. and Singh, B. (2009) 'Performance analysis of 32-bit array multiplier with a carry save adder and with a carry-look-ahead adder', *International Journal of Recent Trends in Engineering*, Vol. 2, No. 6, pp.83–86.

Swee, K.L.S. and Hiung, L.H. (2012) 'Performance comparison review of 32-bit multiplier designs', *Proceedings of the IEEE 4th International Conference on Intelligent and Advanced Systems (ICIAS)*, 12–14 June, Kuala Lumpur, pp.836–841.

Wallace, C.S. (1964) 'A suggestion for a fast multiplier', *IEEE Transactions on Electronic Computers*, Vol. EC-13, No. 1, pp.14–17.

Yao, A., Li, L. and Sun, M. (2011) 'Design of pipeline multiplier based on modified booth's algorithm and wallace tree', in Shen G. and Huang, X. (Eds.): *Advanced Research on Electronic Commerce, Web Application, and Communication. ECWAC 2011. Communications in Computer and Information Science*, Vol. 143, Springer, Berlin, Heidelberg.

Zhang, X., Boussaid, F. and Bermak, A. (2014) '32 bit × 32 bit multiprecision razor-based dynamic voltage scaling multiplier with operands scheduler', *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 22, No. 4, pp.759–770.