electronics

MDPI

Article

Area-Power-Delay-Efficient Multi-Modulus Multiplier Based on Area-Saving Hard Multiple Generator Using Radix-8 Booth-Encoding Scheme on Field Programmable Gate Array

Chao-Tsung Kuo * and Yao-Cheng Wu

Department of Electrical Engineering, National Quemoy University, Kinmen 89250, Taiwan; garygary538@gmail.com

* Correspondence: ctkuo@nqu.edu.tw; Tel.: +886-82-313562; Fax: +886-82-313569

Abstract: A multi-modulus architecture based on the radix-8 Booth encoding of a modulo $(2^n - 1)$ multiplier, a modulo (2^n) multiplier, and a modulo $(2^n + 1)$ multiplier is proposed in this paper. It uses the original single circuit and shares many common circuit characteristics with a small extra circuit to carry out multi-modulus operations. Compared with a previous radix-4 study, the radix-8 architecture can increase the modulation multiplication encoding selection from three codes to four codes. This reduces the use of partial products from $\lfloor n/2 \rfloor$ to $\lfloor n/3 \rfloor + 1$, but it increases the operation complexity for multiplication by three circuits. A hard multiple generator (HMG) is used to address this problem. Two judgment signals in the multi-modulus circuit can be used to perform three operations of the modulo $(2^n - 1)$ multiplier, modulo (2^n) multiplier, and modulo $(2^n + 1)$ multiplier at the same time. The weighted representation is used to reduce the number of partial products. Compared with previously reported methods in the literature, the proposed approach can achieve better performance by being more area-efficient, being faster, consuming low power, and having a lower area-delay product (ADP) and power-delay product (PDP). With the multi-modulus HMG, the proposed modified architecture can save 34.48–55.23% of hardware area. Compared with previous studies on the multi-modulus multiplier, the proposed architecture can save 22.78–35.46%, 4.12–11.15%, 12.59–24.73%, 27.88–38.88%, and 20.49–27.85% of hardware area, delay time, dissipation power, ADP, and PDP, respectively. Xilinx field programmable gate array (FPGA) Vivado 2019.2 tools and the Verilog hardware description language are used for synthesis and implementation. The Xilinx Artix-7 XC7A35T-CSG324-1 chipset is adopted to evaluate the performance.

Keywords: residue number system; radix-8 Booth encoding; hard multiple generator; multi modulus; field programmable gate array

Citation: Kuo, C.-T.; Wu, Y.-C. Area-Power-Delay-Efficient Multi-Modulus Multiplier Based on Area-Saving Hard Multiple Generator Using Radix-8 Booth-Encoding Scheme on Field Programmable Gate Array. *Electronics* **2024**, *13*, 311. https://doi.org/10.3390/electronics13020311

Academic Editors: Charles Tijus, Kuei-Shu Hsu, Teen-Hang Meen, Po-Lei Lee and Chun-Yen Chang

Received: 6 December 2023
Revised: 5 January 2024
Accepted: 8 January 2024
Published: 10 January 2024

1. Introduction

In recent decades, the residue number system (RNS) [1–6] has been increasingly applied in cryptography [2,7], error correction codes [8], and digital signal processing [3], owing to its carry-free nature and parallel computation. A reduced power consumption, shorter latency, and smaller hardware area can be achieved for applications based on RNS modulation addition [9–13] and multiplication [14–25]. When using the multi-modulus architecture, multiple modulus operations can be performed at the same time. Many common hardware circuits can share in the multi-modulus architecture of modulo $(2^n - 1)$, modulo (2^n), and modulo $(2^n + 1)$ multipliers, owing to the commonality of the modulus and similarity of hardware circuits in the modulo multiplication, so only different modules of the circuit need to be additionally designed, which significantly reduces the circuit area. Diminished-1 representation [9,11,12] and weighted representation [13,25] are the two main representations in the RNS-based modulo multiplier. A weighted representation is adopted in the current work.

The traditional modified Booth-encoded multiplier is also called a radix-4 Booth-encoded multiplier [20,25,26], which uses a three-code interpretation. A 0 is added after the least significant bit (LSB), and a 0 is also added in front of the most significant bit (MSB) of the multiplier, which then encodes it in groups of three bits, so that only $\lfloor n/2 \rfloor$ are needed for partial products with n bits. This leads to a reduction in the use of full adders and greatly reduces the circuit area and delay time. Compared with the previous radix-4 research, the radix-8 [20,22–24] architecture can increase the modulation multiplication encoding selection from a three-code to a four-code interpretation, which reduces the use of partial products from $\lfloor n/2 \rfloor$ to $\lfloor n/3 \rfloor + 1$. As the three-code interpretation (radix-4 multiplier) increases to a four-code interpretation (radix-8 multiplier), the partial product is reduced from half of the traditional multiplier to one-third, which further improves the circuit area and delay time. The radix-4-based multiplicand in the three-code interpretation is only multiplication by 1 ($\times 1$) and multiplication by 2 ($\times 2$). Through the carry-free principle in the RNS, the multiplication by 2 ($\times 2$) multiplicand only needs to return the original multiplicand once (shift left by one bit). However, for the radix-8 multiplier, there will be an additional multiplication by 3 ($\times 3$) and multiplication by 4 ($\times 4$) operations to be processed. The multiplication by 4 ($\times 4$) operation can use the same carry-free principle in the RNS to return twice (shift left by 2 bits). However, multiplication by 3 ($\times 3$), which is processed by the hard multiple generator (HMG), needs to be obtained by adding multiplication by 1 ($\times 1$) and multiplication by 2 ($\times 2$) of the original multiplicand. This increases the cost of the hardware area, delay, and power consumption. Therefore, simplifying the HMG for a triple operation is very important. An area-saving modified multi-modulus HMG is first presented for this proposed multi-modulus multiplier.

The proposed architecture of the multi-modulus multiplier based on an area-saving HMG using a radix-8 Booth-encoding scheme can achieve significant improvements in hardware cost, delay time, and power consumption. The structure of the area-delay-power-efficient multi-modulus multiplier proposed in this paper can operate the modulo $(2^n - 1)$, modulo (2^n), and modulo $(2^n + 1)$ multipliers at the same time with only two control signals sharing the same hardware structure. The proposed multi-modulus HMG circuit and modular multiplication can also greatly reduce the hardware cost compared to that of Rama's [20] method. For FPGA implementation, there are many FPGA families and many manufacturers. The propagation time in the LUT (Look-Up Table)/ALM (Adaptive Logic Module) array is different in Xilinx Artix-7, Xilinx Spartan-7, Xilinx Kintex-7, Intel Cyclone-10, and so on. In the proposed work, the Xilinx Artix-7 XC7A35T-CSG324-1 chipset is adopted to evaluate the performance.

The rest of this paper is organized as follows. The methods reported in the literature are described in Section 2. Section 3 presents the proposed multi-modulus HMG and radix-8 Booth-encoding-based multi-modulus multiplier design, which is area-delay-power efficient. The results of the proposed scheme in comparison with those of various other methods are presented in Section 4. Finally, Section 5 concludes the study.

2. Previous Work

2.1. Radix-8 Multi-Modulus Multiplier in $\{2^n - 1, 2^n, 2^n + 1\}$

A structure in which a multi-modulus multiplier can be operated under the same hardware architecture has been reported [20]. This design can greatly reduce the area used. There are three types of modulus multiplication, namely, modulo $(2^n - 1)$, modulo (2^n), and modulo $(2^n + 1)$ multipliers, which can be processed using two control signals. Let X be the multiplicand, Y the multiplier and Z the binary product. Weighted representation is used for modulo m, $m = 2^n - 1$, or 2^n; diminish-1 representation is used for $m = 2^n + 1$, where m is the modulo parameter. The general expression is as follows [20]:

$$|Z|_m = \begin{cases} |X \cdot Y|_m & if\ m = 2^n - 1 \text{ or } 2^n \\ |X \cdot Y + X + Y|_m & if\ m = 2^n + 1 \end{cases} \quad (1)$$

where $|X \cdot Y|_m$ is denoted as the modulo m residue of $X \cdot Y$.

The partial product (PP) can be obtained after taking radix-8 operations of X and Y. The related equation is expressed as [20]:

$$|Z|_m = \begin{cases} \left| \sum_{i=0}^{\lfloor n/3 \rfloor} PP_i \right|_m & if\ m = 2^n - 1 \\ \left| \sum_{i=0}^{\lfloor n/3 \rfloor} PP_i + \sum_{i=0}^{\lfloor n/3 \rfloor} K_i \right|_m & if\ m = 2^n \\ \left| \sum_{i=0}^{\lfloor n/3 \rfloor} PP_i + \sum_{i=0}^{\lfloor n/3 \rfloor} K_i + X + Y \right|_m & if\ m = 2^n + 1 \end{cases} \quad (2)$$

where K_i is the extra compensation parameter. The complete equation of K_i, where KD_i is a dynamic bias and KS_i is a static bias, is as follows [20]:

$$\begin{aligned} \sum_{i=0}^{\lfloor n/3 \rfloor} K_i = & \sum_{i=0}^{\lfloor n/3 \rfloor} \underbrace{2^{3i}\left(\overline{m2_i + m4_i}\right) + \left(\overline{m3_i + m4_i}\right) \cdot 2^{3i+1} + 2^{3i+1} \cdot s_i +}_{KD_i} \\ & \sum_{i=0}^{\lfloor n/3 \rfloor} \underbrace{((m2_i + m4_i) \cdot s_i) 2^{3i+1} + ((m3_i + m4_i) \cdot s_i) \cdot 2^{3i+2}}_{KD_i} \\ & + \sum_{i=0}^{\lfloor n/3 \rfloor} \underbrace{-2^{3i} - 2^{3i+1}}_{KD_i} \underbrace{-2^{3i} + 1}_{KS_i} \end{aligned} \quad (3)$$

where m_{2i}, m_{3i}, m_{4i} denote the ith partial product row of multiplication by 2 bits, multiplication by 3 bits, and multiplication by 4 bits, respectively.

The value of the carry bit (c_i) for an even carry bit (Equation (4)) and an odd bit (Equation (5)) are given by [19]:

$$c_i = \begin{cases} (g_i{}^*, p_i{}^*) \bullet (g_{i-2}{}^*, p_{i-2}{}^*) \bullet \ldots \bullet (g_0{}^*, p_0{}^*) \bullet (g_{n-2}{}^*, p_{n-2}{}^*) \bullet \ldots \bullet (g_{i+2}{}^*, p_{i+2}{}^*); & if\ m = 2^n - 1 \\ (g_i{}^*, p_i{}^*) \bullet (g_{i-2}{}^*, p_{i-2}{}^*) \bullet \ldots \bullet (g_0{}^*, p_0{}^*) \bullet (0,0) \bullet \ldots \bullet (0,0); & if\ m = 2^n \\ (g_i{}^*, p_i{}^*) \bullet (g_{i-2}{}^*, p_{i-2}{}^*) \bullet \ldots \bullet (g_0{}^*, p_0{}^*) \bullet (\overline{p_{n-2}}{}^*, \overline{g_{n-2}}{}^*) \bullet \ldots \bullet (\overline{p_{i+2}}{}^*, \overline{g_{i+2}}{}^*); & if\ m = 2^n + 1 \end{cases} \quad (4)$$

and:

$$c_i = \begin{cases} (g_i{}^*, p_i{}^*) \bullet (g_{i-2}{}^*, p_{i-2}{}^*) \bullet \ldots \bullet (g_1{}^*, p_1{}^*) \bullet (g_{n-1}{}^*, p_{n-1}{}^*) \bullet \ldots \bullet (g_{i+2}{}^*, p_{i+2}{}^*); & if\ m = 2^n - 1 \\ (g_i{}^*, p_i{}^*) \bullet (g_{i-2}{}^*, p_{i-2}{}^*) \bullet \ldots \bullet (g_1{}^*, p_1{}^*) \bullet (0,0) \bullet \ldots \bullet (0,0); & if\ m = 2^n \\ (g_i{}^*, p_i{}^*) \bullet (g_{i-2}{}^*, p_{i-2}{}^*) \bullet \ldots \bullet (g_1{}^*, p_1{}^*) \bullet (\overline{p_{n-1}}{}^*, \overline{g_{n-1}}{}^*) \bullet \ldots \bullet (\overline{p_{i+2}}{}^*, \overline{g_{i+2}}{}^*); & if\ m = 2^n + 1 \end{cases} \quad (5)$$

respectively, where (g_i*, p_i*) is defined as a modified generated–propagated bit pair and (g_i*, p_i*)•(g_j*, p_j*) = (g_i*+ p_i* g_j*, p_i* p_j*).

For the final adder of this study, a Sklansky-based parallel prefix adder [13] is used. The study presents multi-modulus modulo ($2^n - 1$), modulo (2^n), and modulo ($2^n + 1$) multipliers [20] that can reuse the same hardware resources. Nevertheless, the performance of the hardware area, latency, and power consumption still have room for improvement. The improved method and hardware structure are discussed in the next section.

2.2. Hard Multiple Generators

This subsection discusses the HMG for the modulo ($2^n - 1$) [22], modulo (2^n) [23], and modulo ($2^n + 1$) [24] multipliers in the literature. In the Booth encoder (BE), the radix-8 Booth-encoding operation, which can reduce the number of partial products to $\lfloor n/3 \rfloor + 1$ items by means of a four-bit interpretation of the multiplier; multiplication by 1 ($\times$1); multiplication by 2 ($\times$2); multiplication by 3 ($\times$3); multiplication by 4 ($\times$4); and the sign signal is obtained after the operation. Multiplications by 1 ($\times$1), 2 ($\times$2), and 4 ($\times$4) are easy to handle, as multiplications by 2 ($\times$2) and 4 ($\times$4) only need to shift one bit and two bits to the left, respectively. However, multiplication by 3 ($\times$3) is difficult to handle

and cannot be obtained directly from the multiplicand, so the HMG unit is used to operate the process.

There are two processing methods; the first is $|+X|_m + |+2X|_m$, and the second is $|-X|_m + |+4X|_m$, where X is the multiplicand and $|X|_m$ is defined as the modulo operation of X. The first type is clearly better than the second type because the first one does not need to process the 1's compliment operation. The related derivation results of the reported HMG are as follows. The representation of multiplication by 3 ($\times 3$) is as follows:

$$\begin{array}{l} |+3X|_m = |+X|_m + |+2X|_m; \\ |+X|_m = (x_{n-1}x_{n-2}\ldots x_0); \\ |+2X|_m = \begin{cases} (x_{n-2}x_{n-3}\ldots x_0x_{n-1}), if\ m = 2^n - 1 \\ (x_{n-2}x_{n-3}\ldots x_00), if\ m = 2^n \end{cases}; \end{array} \quad (6)$$

The generated bit, propagated bit, half sum bit, and delay half (DH) sum bit are defined as g_i, p_i, h_i, and dh_i, respectively [22–24]:

$$\begin{array}{l} g_i = x_i \cdot\ x_{i-1} \\ p_i = x_i + x_{i-1} \\ h_i = x_i \oplus x_{i-1} \\ dh_i = x_{2i+1} \oplus x_{2i}h_{2i} \end{array} \quad (7)$$

The equation for the carry bit at the odd position is shown as [22–24]:

$$c_{2i-1} = {P_{2i-1}}^{*}{H_{2i-1}}^{**} \quad (8)$$

where ${P_{2i-1}}^{*}$ is a modified propagated bit, and ${H_{2i-1}}^{**}$ is a modified Ling bit [22–24]. The general equation of the modified Ling bit ${H_{2i-1}}^{**}$ is represented as [22–24]:

$${H_{2i-1}}^{**} = ({G_{2i-1}}^{**}, {P_{2i-3}}^{**})\bullet({G_{2i-5}}^{**}, {P_{2i-7}}^{**})\bullet\ldots\bullet({G_{2i-9}}^{**}, {P_{2i-11}}^{**})\bullet\ldots \quad (9)$$

where ${G_{2i-1}}^{**}$ and ${P_{2i-1}}^{**}$ are modified ${G_{2i-1}}^{*}$ and modified ${P_{2i-1}}^{*}$ bits, respectively. ${H_{2i-1}}^{**}$ is used to produce odd carry bits in the HMG and perform HMG prefix operations between ${G_{2i-1}}^{**}$ and ${H_{2i-1}}^{**}$. ${G_{2i-1}}^{**}$ and ${P_{2i-1}}^{**}$ are used to perform the logic OR operation and logic AND operation for the modified generated bit (${G_{2i-1}}^{*}$) and modified propagated bit (${P_{2i-1}}^{*}$), respectively. The modified generated bit (${G_{2i-1}}^{*}$) and modified propagated bit (${P_{2i-1}}^{*}$) are calculated from the generated bit and propagated bit, respectively. ${H_{2i-1}}^{**}$, ${G_{2i-1}}^{**}$, and ${P_{2i-1}}^{**}$ are the intermediate processing units in the HMG operation and can be used to produce the hard multiple bit. The final equation for the sum of bits at the even and odd positions in the HMG is as follows [22–24]:

$$\begin{array}{l} s_{2i} = h_i \oplus ({P_{2i-1}}^{*}{H_{2i-1}}^{**}) \\ s_{2i+1} = dh_i \oplus (h_{2i} \oplus ({P_{2i-1}}^{*}{H_{2i-1}}^{**})) \end{array} \quad (10)$$

From the above derivation of Equations (6)–(10), the block diagram is HMG was presented [22–24].

The proposed multi-modulus HMG structure for three types of modulo $(2^n - 1)$, modulo (2^n), and modulo $(2^n + 1)$ multipliers based on radix-8 operation using the same hardware circuit is presented in the next section.

3. Proposed Multi-Modulus Multiplier Based on Radix-8 Booth Encoding

Figure 1 shows a block diagram of the system architecture of the proposed multi-modulus multiplier based on an area-saving HMG using a radix-8 Booth-encoding scheme. Multi-modulus multipliers are defined to support modulo $(2^n - 1)$, modulo (2^n), and modulo $(2^n + 1)$ multiplication functions in the same circuit hardware by the control signal (S1, S0). When (S1, S0) = (0, 0), the modulo $(2^n - 1)$ multiplier operation is selected; when (S1, S0) = (0, 1), the modulo (2^n) multiplier operation is selected; and when (S1, S0) = (1, 0),

the modulo (2^n + 1) multiplier operation is selected. In Figure 1, the proposed multi-modulus multiplier includes the Booth encoder (BE) unit, hard multiple generator (HMG) unit, Booth selector (BS) unit, compensation unit, an inverse end-around-carry carry-save adder tree (IEAC CSA tree), and the proposed improved parallel prefix adder unit. The multiplier is Booth-encoded by 4 bits to generate ×1, ×2, ×3, ×4, and s signals. Such an encoding can reduce the number of partial products. The multiplicand, +2X (one left shift), +4X (two left shift), and +3X are generated by the HMG. They then enter the BS unit and are selected by the output of the Booth encoder and obtain the output of the *i*th-row partial product (*pp*). Afterwards, the partial product (*pp*) and compensation value C1 and C2 from the compensation circuit are fed into the IEAC CSA tree and summed to obtain sum (S) and carry (C). Finally, the obtained S and C are summed through the final parallel prefix adder to obtain the product O. The proposed multi-modulus HMG and proposed radix-8 multi-modulus multiplier are discussed in the following subsection.

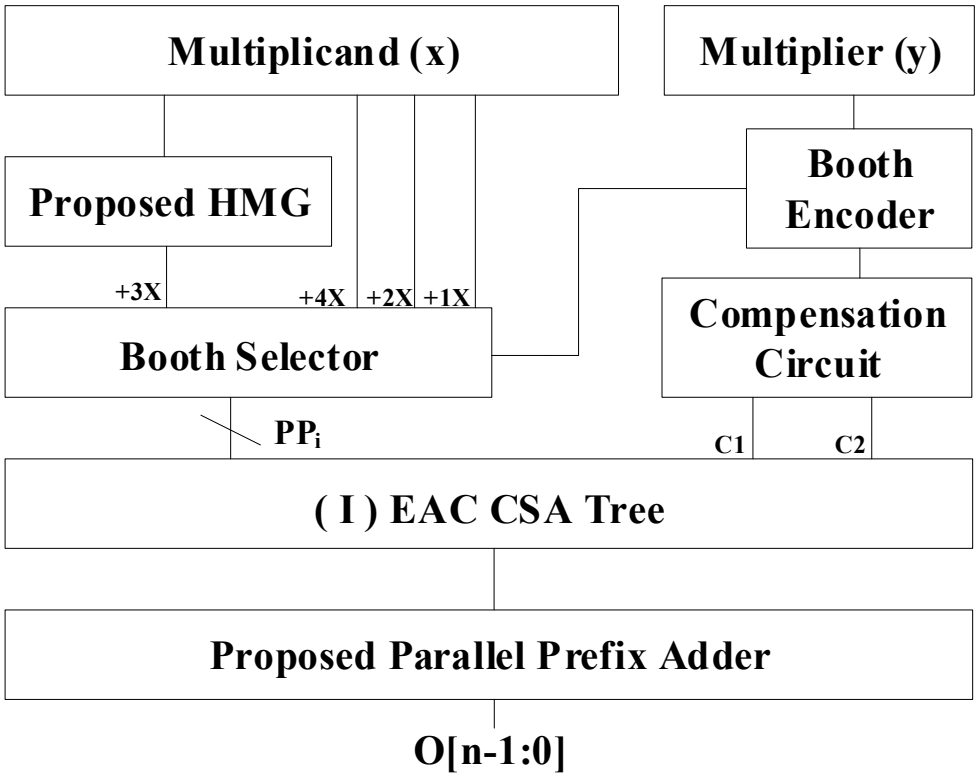

Figure 1. Block diagram of the proposed multi-modulus multiplier.

3.1. Proposed Multi-Modulus Hard Multiple Generator

In this subsection, the modified multi-modulus HMG for the modulo ($2^n - 1$), modulo (2^n), and modulo (2^n + 1) multiplier operations is discussed. The proposed structure of radix-8 multi-modulus HMG (n = 8) is designed as shown in Figure 2. The proposed structure includes a GP**P* block, DH block, SM1, SM2, prefix operator unit (grey circle), and post-processing unit (grey square, white square, grey diamond, and white diamond).

In Figures 3 and 4, SM1 and SM2 refer to the special multiplexer 1 and special multiplexer 2, respectively. These blocks are used to generate different input signals from the multi-modulus by selecting (S1, S0).

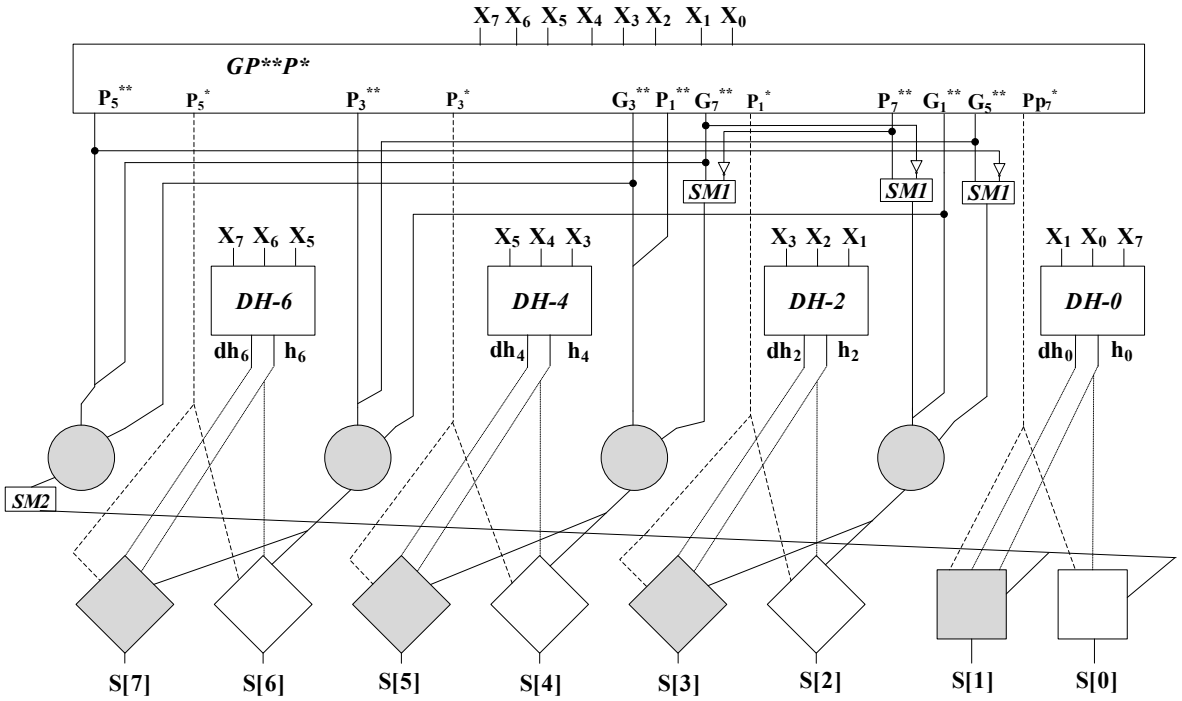

Figure 2. Proposed structure of the radix-8 multi-modulus hard multiple generator (8-bit).

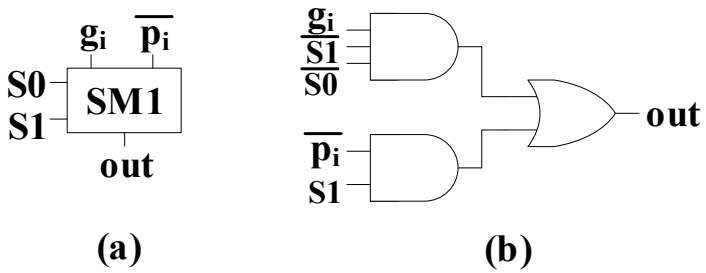

Figure 3. (**a**) Block diagram of the proposed SM1. (**b**) Inner circuit of SM1.

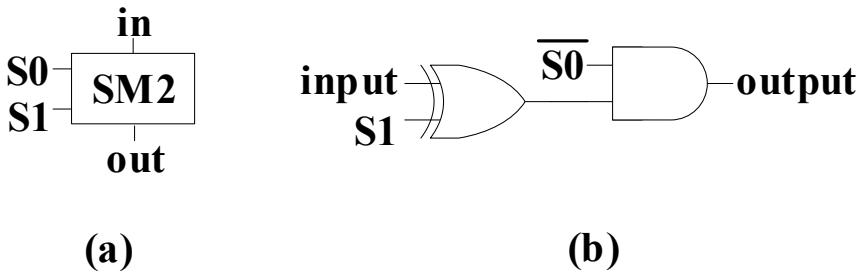

Figure 4. (**a**) Block diagram of the proposed SM2. (**b**) Inner circuit of SM2.

In the block diagram of the *GP***P** function, X_i is the input of the multiplicand, ${G_i}^*$ and ${P_i}^*$ are, respectively, the modified generated and propagated bits in the HMG, and ${G_i}^{**}$ and ${P_i}^{**}$ are, respectively, the modified ${G_i}^*$ and ${P_i}^*$ bits. The related equations of ${G_i}^{**}$, ${P_i}^{**}$, ${G_i}^*$, and ${P_i}^*$ are derived from the modulo $(2^n - 1)$ multiplier [22], modulo (2^n) multiplier [23], and modulo $(2^n + 1)$ [24] multiplier:

$$\begin{cases} {G_i}^{**} = {G_i}^* + {G_{i-2}}^* \text{ , } if\ i = 1 \text{ , } for\ m = 2^n - 1 \\ {G_i}^{**} = {G_i}^* + 0 \text{ , } if\ i = 1 \text{ , } for\ m = 2^n \\ {G_i}^{**} = {G_i}^* + \overline{{P_{i-2}}^*} \text{ , } if\ i = 1 \text{ , } for\ m = 2^n + 1 \end{cases} \tag{11}$$

$$\begin{cases} P_i{}^{**} = P_i{}^{*} P_{i-2}{}^{*} \text{ , } if\ i = 1 \text{ , } for\ m = 2^n - 1 \\ P_i{}^{**} = P_i{}^{*} \cdot 0 \text{ , } if\ i = 1 \text{ , } for\ m = 2^n \\ P_i{}^{**} = P_i{}^{*} \overline{G_{i-2}}{}^{*} \text{ , } if\ i = 1 \text{ , } for\ m = 2^n + 1 \end{cases} \tag{12}$$

$$\begin{cases} G_i{}^{**} = G_i{}^{*} + G_{i-2}{}^{*} \text{ , } if\ 1 < i < n \text{ , } for\ m = 2^n - 1 \\ G_i{}^{**} = G_i{}^{*} + G_{i-2}{}^{*} \text{ , } if\ 1 < i < n \text{ , } for\ m = 2^n \\ G_i{}^{**} = G_i{}^{*} + G_{i-2}{}^{*} \text{ , } if\ 1 < i < n \text{ , } for\ m = 2^n + 1 \end{cases} \tag{13}$$

$$\begin{cases} P_i{}^{**} = P_i{}^{*} P_{i-2}{}^{*} \text{ , } if\ 1 < i < n \text{ , } for\ m = 2^n - 1 \\ P_i{}^{**} = P_i{}^{*} P_{i-2}{}^{*} \text{ , } if\ 1 < i < n \text{ , } for\ m = 2^n \\ P_i{}^{**} = P_i{}^{*} P_{i-2}{}^{*} \text{ , } if\ 1 < i < n \text{ , } for\ m = 2^n + 1 \end{cases} \tag{14}$$

From Equation (11) to Equation (14), when $i = 1$, $G_i{}^{*}$ and $P_i{}^{*}$ can be rewritten as:

$$\begin{cases} G_1{}^{*} = x_0 \cdot (x_1 + x_{-1}) \text{ , } for\ m = 2^n - 1 \\ G_1{}^{*} = x_0 \cdot (x_1 + 0) \text{ , } for\ m = 2^n \\ G_1{}^{*} = x_0 \cdot (x_1 + \overline{x_{-1}}) \text{ , } for\ m = 2^n + 1 \end{cases} \tag{15}$$

$$\begin{cases} P_1{}^{*} = x_0 + (x_1 \cdot x_{-1}) \text{ , } for\ m = 2^n - 1 \\ P_1{}^{*} = x_0 + (x_1 \cdot 0) \text{ , } for\ m = 2^n \\ P_1{}^{*} = x_0 + (x_1 \cdot \overline{x_{-1}}) \text{ , } for\ m = 2^n + 1 \end{cases} \tag{16}$$

From the above definitions of $G_i{}^{*}$, $P_i{}^{*}$, $G_i{}^{**}$, and $P_i{}^{**}$ for the modulo $(2^n - 1)$, modulo (2^n), and modulo $(2^n + 1)$ multipliers, the block diagram of the proposed *GP****P** function is shown in Figure 5. The $Pp_7{}^{*}$ signal is used to select the $P_7{}^{*}$, 0, or $\overline{P_7{}^{*}}$ signals for the modulo $(2^n - 1)$, modulo (2^n), or modulo $(2^n + 1)$ multipliers, respectively.

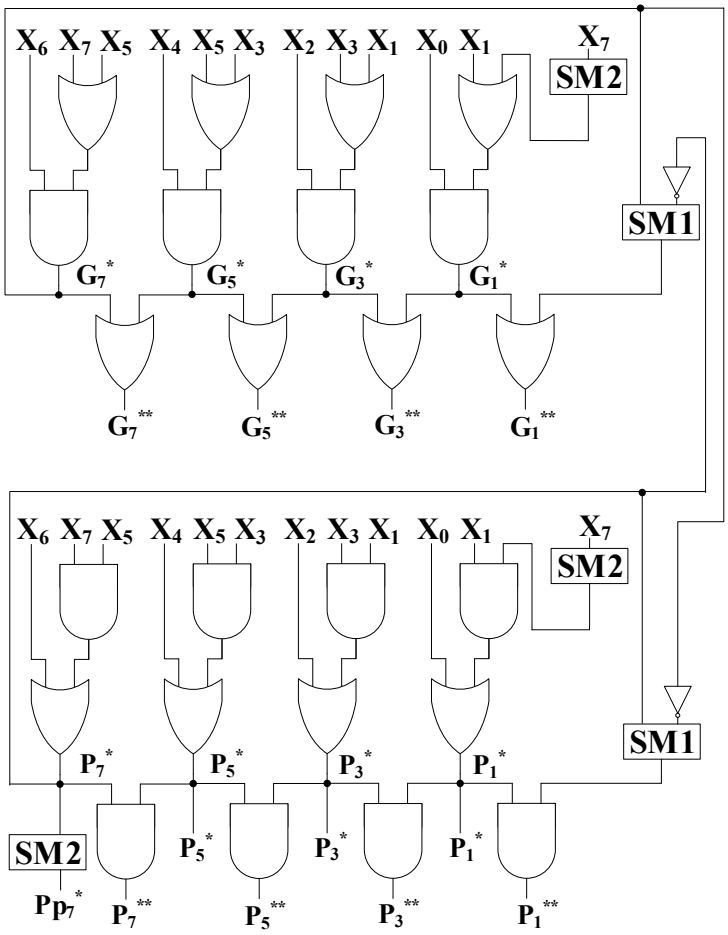

Figure 5. Proposed block diagram of the *GP****P** function for $n = 8$.

For the DH block, multiplication by 2 ($\times 2$) for the modulo $(2^n - 1)$, modulo (2^n), and modulo $(2^n + 1)$ multipliers is expressed as:

$$|+2X|_m = \begin{cases} (x_{n-2}x_{n-3}\ldots x_0x_{n-1}) & if\ m = 2^n - 1 \\ (x_{n-2}x_{n-3}\ldots x_00) & if\ m = 2^n \\ (x_{n-2}x_{n-3}\ldots x_0\overline{x_{n-1}}) & if\ m = 2^n + 1 \end{cases}. \quad (17)$$

Based on Equations (6) and (7) and Equation (17), the DH component is as shown in Figure 6. In Figure 6a, $i = 0$ is shown for the end-around-carry bit in the multi-modulus. Figure 6b is the general circuit implementation for $i > 0$.

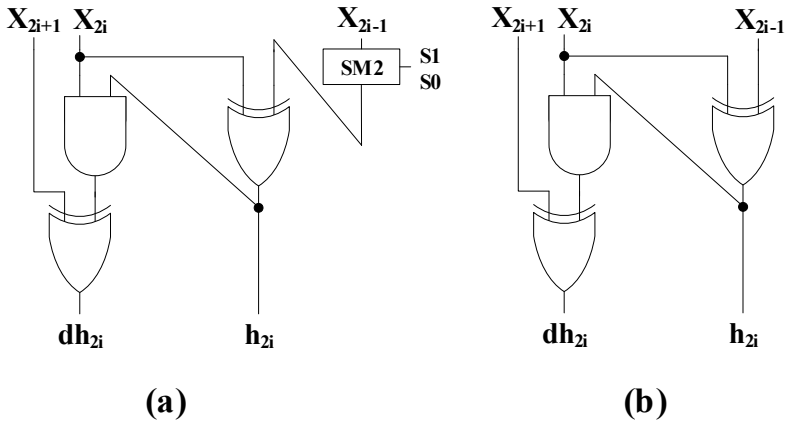

Figure 6. (**a**) DH $- i$ for $i = 0$, and (**b**) DH $- i$ for $0 < i < n$, where i is even.

In the prefix operator block, ${H_{2i-1}}^{**}$ (i = 1, 2, 3,...) is the modified Ling bit and is expressed at odd positions, which is defined as $H^{**}_{2i-1} = (G^{**}_{2i-1}, P^{**}_{2i-3})\bullet(G^{**}_{2i-5}, P^{**}_{2i-7})$, where $H^{**}_{-1} = H^{**}_{n-1}$, $G^{**}_{-i} = G^{**}_{n-i}$, and $P^{**}_{-i} = P^{**}_{n-i}$ [22–24]. Taking n = 8 as an example, ${H_{2i-1}}^{**}$ can be shown as Equation (18). The index of ${H_{2i-1}}^{**}$ at position 1 and position 5 is different from the index of ${H_{2i-1}}^{**}$ at position 3 and position 7. Therefore, ${H_{2i-1}}^{**}$ is separated into two groups: ${H_{4k+1}}^{**}$ and ${H_{4k+3}}^{**}$ [24], where k = 0, 1, 2, 3, ... That is to say, ${H_{2i-1}}^{**} = ({H_1}^{**}, {H_3}^{**}, {H_5}^{**}, {H_7}^{**}, {H_9}^{**}, {H_{11}}^{**}\ldots)$ is divided into two groups: ${H_{4k+1}}^{**} = ({H_1}^{**}, {H_5}^{**}, {H_9}^{**}\ldots)$ and ${H_{4k+3}}^{**} = ({H_3}^{**}, {H_7}^{**}, {H_{11}}^{**}\ldots)$. The general expressions of ${H_{2i-1}}^{**}$ for modulo $(2^n - 1)$, modulo (2^n), and modulo $(2^n + 1)$ are derived from the modulo $(2^n - 1)$ multiplier [22], modulo (2^n) multiplier [23], and modulo $(2^n + 1)$ multiplier [24], respectively:

$$\begin{aligned} H_1^{**} &= (G_1^{**}, \overline{G}_7^{**})\bullet(\overline{P}_5^{**}, \overline{G}_3^{**}) \\ H_3^{**} &= (G_3^{**}, P_1^{**})\bullet(\overline{P}_7^{**}, \overline{G}_5^{**}) \\ H_5^{**} &= (G_5^{**}, P_3^{**})\bullet(G_1^{**}, \overline{G}_7^{**}) \\ H_7^{**} &= (G_7^{**}, P_5^{**})\bullet(G_3^{**}, P_1^{**}) \end{aligned} \quad (18)$$

$$\begin{cases} {H_{4k+1}}^{**} = \underbrace{({G_{4k+1}}^{**}, {P_{4k-1}}^{**})\bullet\ldots\bullet({G_1}^{**}, {P_{-1}}^{**})\bullet\ldots\bullet({G_{4k-3}}^{**}, {P_{4k-5}}^{**})}_{\frac{n}{4}} \\ {H_{4k+3}}^{**} = \underbrace{({G_{4k+3}}^{**}, {P_{4k+1}}^{**})\bullet\ldots\bullet({G_3}^{**}, {P_1}^{**})\bullet\ldots\bullet({G_{4k-1}}^{**}, {P_{4k-3}}^{**})}_{\frac{n}{4}} \end{cases}, for \text{ modulo } (2^n - 1) \quad (19)$$

$$\begin{cases} {H_{4k+1}}^{**} = \underbrace{({G_{4k+1}}^{**}, {P_{4k-1}}^{**})\bullet\ldots\bullet({G_1}^{**}, 0)\bullet\ldots\bullet(0, 0)}_{\frac{n}{4}} \\ {H_{4k+3}}^{**} = \underbrace{({G_{4k+1}}^{**}, {P_{4k-1}}^{**})\bullet\ldots\bullet({G_3}^{**}, {P_1}^{**})\bullet\ldots\bullet(0, 0)}_{\frac{n}{4}} \end{cases}, for \text{ modulo } 2^n \quad (20)$$

$$\begin{cases} H_{4k+1}{}^{**} = \underbrace{(G_{4k+1}{}^{**}, P_{4k-1}{}^{**})\bullet \ldots \bullet \left(G_1{}^{**}, \overline{G_{-1}}{}^{**}\right)\bullet \ldots \bullet \left(\overline{P_{4k-3}}{}^{**}, \overline{G_{4k-5}}{}^{**}\right)}_{\frac{n}{4}} \\ H_{4k+3}{}^{**} = \underbrace{(G_{4k+3}{}^{**}, P_{4k+1}{}^{**})\bullet \ldots \bullet (G_3{}^{**}, P_1{}^{**})\bullet \ldots \bullet \left(\overline{P_{4k-1}}{}^{**}, \overline{G_{4k-3}}{}^{**}\right)}_{\frac{n}{4}} \end{cases}, \; for \text{ modulo}(2^n+1) \tag{21}$$

From Equation (9) and the description of $H_{2i-1}{}^{**}$ above, the relative logic circuit is obtained as shown in Figure 7.

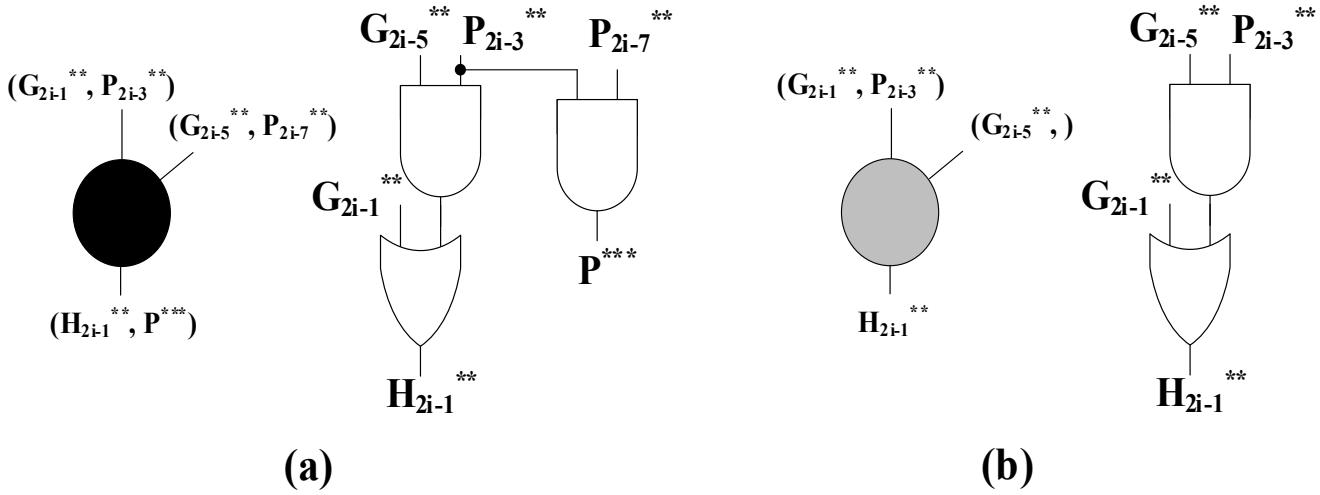

Figure 7. (**a**) Prefix operator (HP) and (**b**) prefix operator (H) [24].

For the post-processing unit, the block diagrams of the grey square, white square, grey diamond, and white diamond are shown in Figures 8 and 9. For $i = 0$, the circuit implementation of the even-position sum bit and odd-position sum bit is designed as shown in Figure 8a,b, respectively. For $i > 0$, the circuit implementation of the even-position sum bit and odd-position sum bit is designed as shown in Figure 9a,b, respectively.

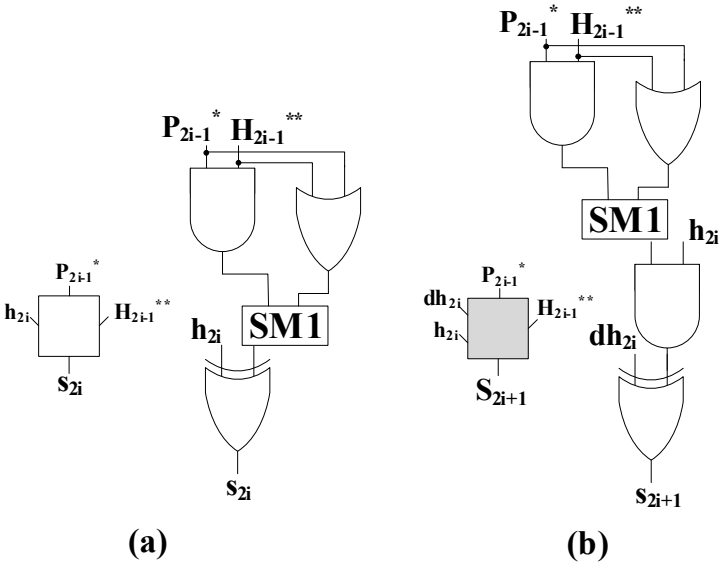

Figure 8. For $i = 0$, the (**a**) even-position sum bit and (**b**) odd-position sum bit.

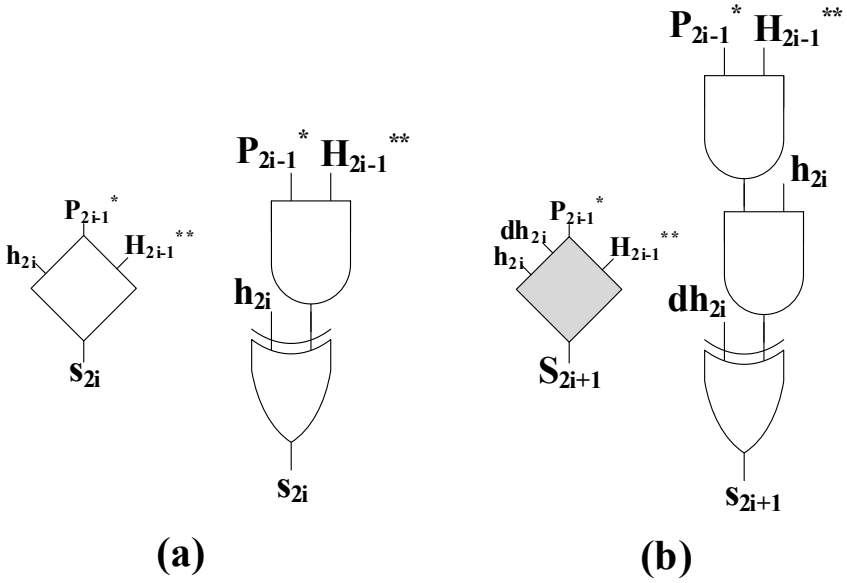

Figure 9. For $i > 0$, the (**a**) even-position sum bit and (**b**) odd-position sum bit.

The final results of the HMG for the sum bit are expressed as follows. The equations for the even-position sum bit and odd-position sum bit for $i = 0$ are expressed as Equations (22) and (24), respectively, and the equations for the even-position sum bit and odd-position sum bit for $i > 0$ are expressed as Equations (23) and (25), respectively:

$$\begin{cases} s_{2i} = h_{2i} \oplus ({P_{2i-1}}^{*} {H_{2i-1}}^{**}),\ if\ i = 0\ , for\ m = 2^n - 1 \\ s_{2i} = h_{2i} \oplus 0, \quad if\ i = 0\ , for\ m = 2^n \\ s_{2i} = h_{2i} \oplus \left(\overline{P_{2i-1}}^{*} + \overline{H_{2i-1}}^{**}\right),\ if\ i = 0\ , for\ m = 2^n + 1 \end{cases} \tag{22}$$

$$\begin{cases} s_{2i} = h_{2i} \oplus ({P_{2i-1}}^{*} {H_{2i-1}}^{**}),\ if\ 0 < i < n/2\ , for\ m = 2^n - 1 \\ s_{2i} = h_{2i} \oplus ({P_{2i-1}}^{*} {H_{2i-1}}^{**}),\ if\ 0 < i < n/2\ , for\ m = 2^n \\ s_{2i} = h_{2i} \oplus ({P_{2i-1}}^{*} {H_{2i-1}}^{**}),\ if\ 0 < i < n/2\ , for\ m = 2^n + 1 \end{cases} \tag{23}$$

$$\begin{cases} s_{2i+1} = dh_i \oplus ({P_{2i-1}}^{*} {H_{2i-1}}^{**}) h_{2i},\ if\ i = 0\ , for\ m = 2^n - 1 \\ s_{2i+1} = dh_i \oplus 0 \cdot h_{2i}, \quad if\ i = 0\ , for\ m = 2^n \\ s_{2i+1} = dh_i \oplus \left(\overline{P_{2i-1}}^{*} + \overline{H_{2i-1}}^{**}\right) h_{2i},\ if\ i = 0\ , for\ m = 2^n + 1 \end{cases} \tag{24}$$

$$\begin{cases} s_{2i+1} = dh_i \oplus ({P_{2i-1}}^{*} {H_{2i-1}}^{**}) h_{2i},\ if\ 0 < i < n/2\ , for\ m = 2^n - 1 \\ s_{2i+1} = dh_i \oplus ({P_{2i-1}}^{*} {H_{2i-1}}^{**}) h_{2i},\ if\ 0 < i < n/2\ , for\ m = 2^n \\ s_{2i+1} = dh_i \oplus ({P_{2i-1}}^{*} {H_{2i-1}}^{**}) h_{2i},\ if\ 0 < i < n/2\ , for\ m = 2^n + 1 \end{cases} \tag{25}$$

From the above design of the sub-circuit in the HMG, the proposed structure of the radix-8 multi-modulus HMG (n = 8) can be designed as shown in Figure 2. It should be noted that Equation (11) to Equation (16), Equation (18) to Equation (21), and Equation (22) to Equation (25) are integrated and modified equations from the modulo ($2^n - 1$) [22], modulo (2^n) [23], and modulo ($2^n + 1$) multipliers [24].

3.2. Proposed Radix-8 Multi-Modulus Multiplier

In this subsection, the proposed radix-8 multi-modulus multiplier is discussed. Let X be the multiplicand and Y the multiplier. The modulo m of $X \times Y$ is expressed as $|X \times Y|_m$. Using the representation of radix-8, Y can be expressed as $Y = 2^{3i}(y_{3i-1} + y_{3i} + 2y_{3i+1} - 4y_{3i+2})$, and the modulo m of X × Y can be expressed as:

$$|X \times Y|_m = \left| X \times 2^{3i}(y_{3i-1} + y_{3i} + 2y_{3i+1} - 4y_{3i+2} \right|_m \tag{26}$$

The truth table of the four-codes interpretation based on radix-8 is presented in Table 1 [20]. The multiplication by 1 (×1), multiplication by 2 (×2), multiplication by 3 (×3), multiplication by 4 (×4), and sign signal are obtained from the BE circuit, which is shown in Figure 10 [20]. The BS is designed as shown in Figure 11a based on the corresponding signals from the BE. In order to reduce the gate count of the BS in the $(\lfloor n/3 \rfloor + 1)$th row, when $n = 6k + 4$ and $n = 6k$, where k is a positive integer, the BE of the $\{\lfloor n/3 \rfloor + 1\}$th row is $[Y_{3i-1}\ Y_{3i-2}\ 0\ 0]$ and $[Y_{3i-1}\ 0\ 0\ 0]$, the BS can be redesigned as shown in Figure 11b,c, respectively. The hardware area can be effectively reduced as shown in Figure 11b,c. In the BS block, the input signal is multiplication by 1 (×1), multiplication by 2 (×2), multiplication by 3 (×3), multiplication by 4 (×4), and the sign bit (s). Multiplication by 2 (×2) and multiplication by 4 (×4) shift one bit and two bits of the original signal to the left, respectively. Multiplication by 3 (×3) is produced from the proposed multi-modulus HMG structure. The sign bit is used to produce the positive or negative multiple. The output of the BS block is the partial product (*pp*). The end-around-carry is operated based on modulo $\{2^n - 1,\ 2^n,\ 2^n{+}1\} = \{x_{-1},\ 0,\ \overline{x_{-1}}\}$ regulation. SM2 (S1, S0), as depicted in Figure 4, is used to select the modulo multiplier, which is (S1, S0) = {00, 01, 10} = modulo $\{2^n - 1,\ 2^n,\ 2^n{+}1\} = \{pp,\ 0,\ \overline{pp}\}$. A weighted representation of the system structure is adopted for the proposed modulo $(2^n - 1)$, modulo (2^n), and modulo $(2^n + 1)$ multipliers.

Table 1. Truth table for the proposed radix-8 Booth encoder [20].

$Y_{3i+2}\ Y_{3i+1}\ Y_{3i}\ Y_{3i-1}$		Operation
0000	1111	0
0001	0010	×(+1)
0011	0100	×(+2)
0101	0110	×(+3)
0111		×(+4)
1000		×(−4)
1001	1010	×(−3)
1011	1100	×(−2)
1101	1110	×(−1)

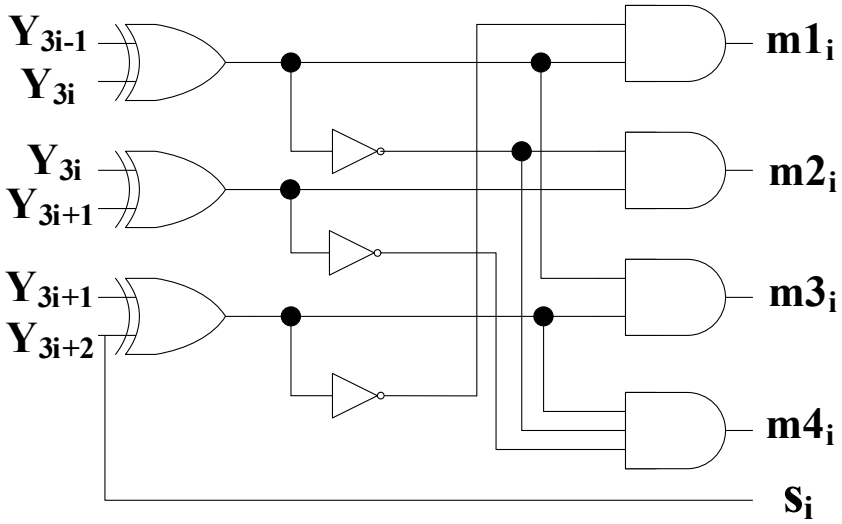

Figure 10. The Booth encoder of the radix-8 Booth-encoding-based architecture [20].

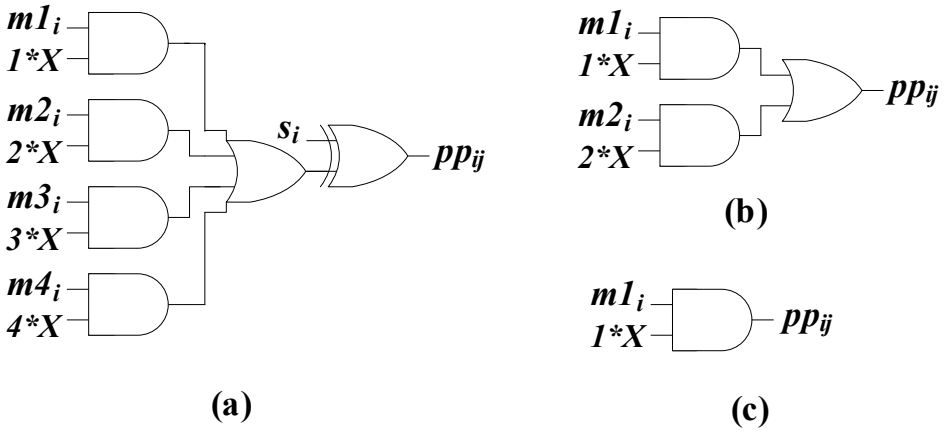

Figure 11. (**a**) The Booth selector of the radix-8 Booth-encoding-based architecture [20]. (**b**) The Booth selector for $n = 6k + 4$. (**c**) The Booth selector for $n = 6k$.

For the modulo $(2^n + 1)$ multiplier, the compensation value is used to compensate for the general output of the partial product. The compensation circuit that produces the compensation value of C1 and C2 in the proposed approach is discussed below. From Equation (3), the compensation for circuit K_i can be rewritten as follows and divided into two parts, denoted as C_1'(the first two rows of the equation) and C_2':

$$\begin{aligned}\sum_{i=0}^{\lfloor n/3 \rfloor} K_i = \quad & \underbrace{\{\sum_{i=0}^{\lfloor n/3 \rfloor} 2^{3i}\left(\overline{m2_i + m4_i}\right) + \left(\overline{m3_i + m4_i}\right) \cdot 2^{3i+1}}_{C_1'} \\ & \underbrace{+ \sum_{i=0}^{\lfloor n/3 \rfloor} s_i(m2_i + m4_i)2^{3i+1} + s_i(m3_i + m4_i) \cdot 2^{3i+2}\}}_{(C_1')} \\ & \underbrace{+\{\sum_{i=0}^{\lfloor n/3 \rfloor} 2^{3i+1} \cdot s_i + \sum_{i=0}^{n/3} -2^{3i} - 2^{3i+1} - 2^{3i} + 1\}}_{C_2'}\end{aligned} \tag{27}$$

From Equation (27), $C_2{}'$ can be rewritten as:

$$C2\prime = \sum_{i=1}^{\lfloor n/3 \rfloor} 2^{3i} \cdot 1 + \sum_{i=0}^{\lfloor n/3 \rfloor} 2^{3i+1} \cdot \overline{s_i} + \sum_{i=0}^{\lfloor n/3 \rfloor - 1} 2^{3i+2} s_i \tag{28}$$

For $C_1{}'$ in Equation (27), the 2^{3i+1} terms can be summed as:

$$\sum_{i=0}^{\lfloor n/3 \rfloor} [\underbrace{\left(\overline{m3_i + m4_i}\right)}_{K1_i} \cdot 2^{3i+1} + \underbrace{s_i(m2_i + m4_i)}_{K2_i} \cdot 2^{3i+1}] \tag{29}$$

where $K1_i$ and $K2_i$ are defined as $\left(\overline{m3_i + m4_i}\right)$ and $s_i(m2_i + m4_i)$, respectively.

$K1_i + K2_i$ can be written as:

$$K1_i + K2_i = (K1_i \oplus K2_i)2^0 + (K1_i \bullet K2_i)2^1 \tag{30}$$

where "$\oplus$" represents the logic Exclusive OR gate, and "$\bullet$" represents the logic AND gate.

The (2^{3i+1})th term of $K1_i + K2_i$ is 0 when $(K1_i, K2_i) = (0, 0)$ or carry out when $(K1_i, K2_i) = (1, 1)$. Therefore, the (2^{3i+1})th term can be rewritten as:

$$\sum_{i=0}^{\lfloor n/3 \rfloor} \underbrace{\left(\overline{m3_i + m4_i}\right)}_{K1_i} \cdot 2^{3i+1} \oplus \underbrace{s_i(m2_i + m4_i)}_{K2_i} \cdot 2^{3i+1} \tag{31}$$

And by merging the (2^{3i+2})th term in Equation (27), it can be rewritten as:

$$\sum_{i=0}^{\lfloor n/3 \rfloor - 1} 2^{3i+2}[((m3_i + m4_i) \cdot s_i) \oplus (K1_i \cdot K2_i)] \tag{32}$$

The exclusive OR logic symbol is used in Equation (32) because $(m3_i + m4_i)$ and $K1_i$ do not appear simultaneously.

In Equation (32), when $n = 3k + 2$ bits for $k = 1, 2, 3, \ldots$, the sum of $K1_i$ and $K2_i$ at the highest bit position in 2^{3i+1} carry out to 2^{3i+2} when $(K1_i, K2_i) = (1, 1)$. It cannot also be represented for n bits. Therefore, for $i = \lfloor n/3 \rfloor$, it should appear at the upper bound of $i = \lfloor n/3 \rfloor$. Taking $n = 8$ as an example, the upper bound of $\lfloor n/3 \rfloor$ is 2. For the (2^7)th bit, the sum of $K1_i$ and $K2_i$ probably carries out to 2^8. Therefore, merging the (2^{3i+1})th term of $C_2{}'$ in Equation (28) for $C_1{}'$ at $i = \lfloor n/3 \rfloor$ yields:

$$\sum_{i=\lfloor n/3 \rfloor}^{\lfloor n/3 \rfloor} 2^{3i+1}\left[\left(\overline{m3_i + m4_i}\right) + ((m2_i + m4_i) \cdot s_i)\right] \tag{33}$$

and for $C_2{}'$ at $i = \lfloor n/3 \rfloor$, it yields:

$$\sum_{i=\lfloor n/3 \rfloor}^{\lfloor n/3 \rfloor} 2^{3i+1}[\overline{s_i} \oplus (K1_i \cdot K2_i)] \tag{34}$$

Here, the weighted representation is adopted to replace the original diminish-1 representation. Therefore, the extra circuit for adding 2 should be processed in the compensation circuit for the modulo $(2^n + 1)$ multiplier. Merging the circuit for adding 2 and $C_2{}'$ in Equation (28), which makes $i = 0$, the modified value is obtained as follows:

$$\begin{aligned} \sum_{i=0}^{0}(2^{3i+1} \cdot \overline{s_i}) + 2 + \sum_{i=0}^{0} 2^{3i+2} s_i &= \sum_{i=0}^{0} 2^{3i+1}(1 + \overline{s_i}) + \sum_{i=0}^{0} 2^{3i+2} s_i \\ &= \sum_{i=0}^{0} 2^{3i+1} \cdot (\overline{s_i} \oplus 1) + \sum_{i=0}^{0} 2^{3i+2}(s_i + 1) \end{aligned} \tag{35}$$

For the modulo (2^n) multiplier, the compensation value is described as follows [20]:

$$C_1 = \sum_{i=0}^{\lfloor n/3 \rfloor} 2^{3i} \cdot s_i \tag{36}$$

According to the derivation in this subsection, for the modulo $(2^n + 1)$ multiplier, the final compensation of C_1 (replacing $C_1{}'$) can be obtained as follows:

$$\begin{aligned} C_1 = & \sum_{i=0}^{\lfloor n/3 \rfloor - 1} 2^{3i+2}[((m3_i + m4_i) \cdot s_i) \oplus (K1_i \cdot K2_i)] + \sum_{i=0}^{\lfloor n/3 \rfloor} 2^{3i}\left(\overline{m2_i + m4_i}\right) \\ & + \begin{cases} \sum_{i=0}^{\lfloor \frac{n}{3} \rfloor} 2^{3i+1}\left[\underbrace{\left(\overline{m3_i + m4_i}\right)}_{K1_i} \oplus \underbrace{((m2_i + m4_i) \cdot s_i)}_{K2_i}\right] \\ \quad , when\ n \neq (3k+2)\ bit, k = 1, 2, 3, \ldots \\ \sum_{i=0}^{\lfloor \frac{n}{3} \rfloor - 1} 2^{3i+1}\left[\underbrace{\left(\overline{m3_i + m4_i}\right)}_{K1_i} \oplus \underbrace{((m2_i + m4_i) \cdot s_i)}_{K2_i}\right] + \sum_{i=\lfloor \frac{n}{3} \rfloor}^{\lfloor \frac{n}{3} \rfloor} 2^{3i+1}\left[\left(\overline{m3_i + m4_i}\right) + ((m2_i + m4_i) \cdot s_i)\right] \\ \quad , when\ n = (3k+2)\ bit, k = 1, 2, 3, \ldots \end{cases} \end{aligned} \tag{37}$$

For modulo $2^n + 1$, the final compensation of C_2 (replacing C_2') is obtained as follows:

$$C_2 = \sum_{i=1}^{\lfloor n/3 \rfloor} 2^{3i} \cdot 1 + \sum_{i=0}^{0} 2^{3i+1} \cdot (\overline{s_i} \oplus 1) + \sum_{i=0}^{0} 2^{3i+2}(s_i + 1) + \sum_{i=1}^{\lfloor n/3 \rfloor - 1} 2^{3i+2} \cdot s_i + \begin{cases} \sum_{i=1}^{\lfloor n/3 \rfloor} 2^{3i+1} \cdot \overline{s_i}, when\ n \neq (3k+2)\ bit, k = 1, 2, 3, \ldots \\ \sum_{i=1}^{\lfloor n/3 \rfloor - 1} 2^{3i+1} \cdot \overline{s_i} + \sum_{i=\lfloor n/3 \rfloor}^{\lfloor n/3 \rfloor} 2^{3i+1}[\overline{s_i} \oplus (K1_i \cdot K2_i)], when n = (3k+2)\ bit, k = 1, 2, 3, \ldots. \end{cases} \quad (38)$$

The final result of $|Z|_m$ can be represented as:

$$|Z|_m = \left\{ \left| \sum_{i=0}^{\lfloor n/3 \rfloor} PP_i + C1 + C2 \right|_m \right. \quad (39)$$

The compensation value C_2 is only needed to compensate for the modulo $(2^n + 1)$ multiplier. Therefore, two input AND gates are used with the selected signal S1 (Mod S1). The compensation value C_1 is needed to compensate for the modulo (2^n) and modulo $(2^n + 1)$ multipliers. The compensation circuit for $n = 8$ is shown in Figure 12. It should be noted that the modulo $(2^n - 1)$ multiplier need not be compensated for by the extra compensation circuit. The final proposed structure of the radix-8 multi-modulus multiplier for 8 bits ($n = 8$) is shown in Figure 13, which includes a partial product unit, IEAC unit, and parallel prefix adder. The Lander–Fisher [12] structure is used for the improved parallel prefix adder circuit, which is shown in Figure 14 ($n = 8$).

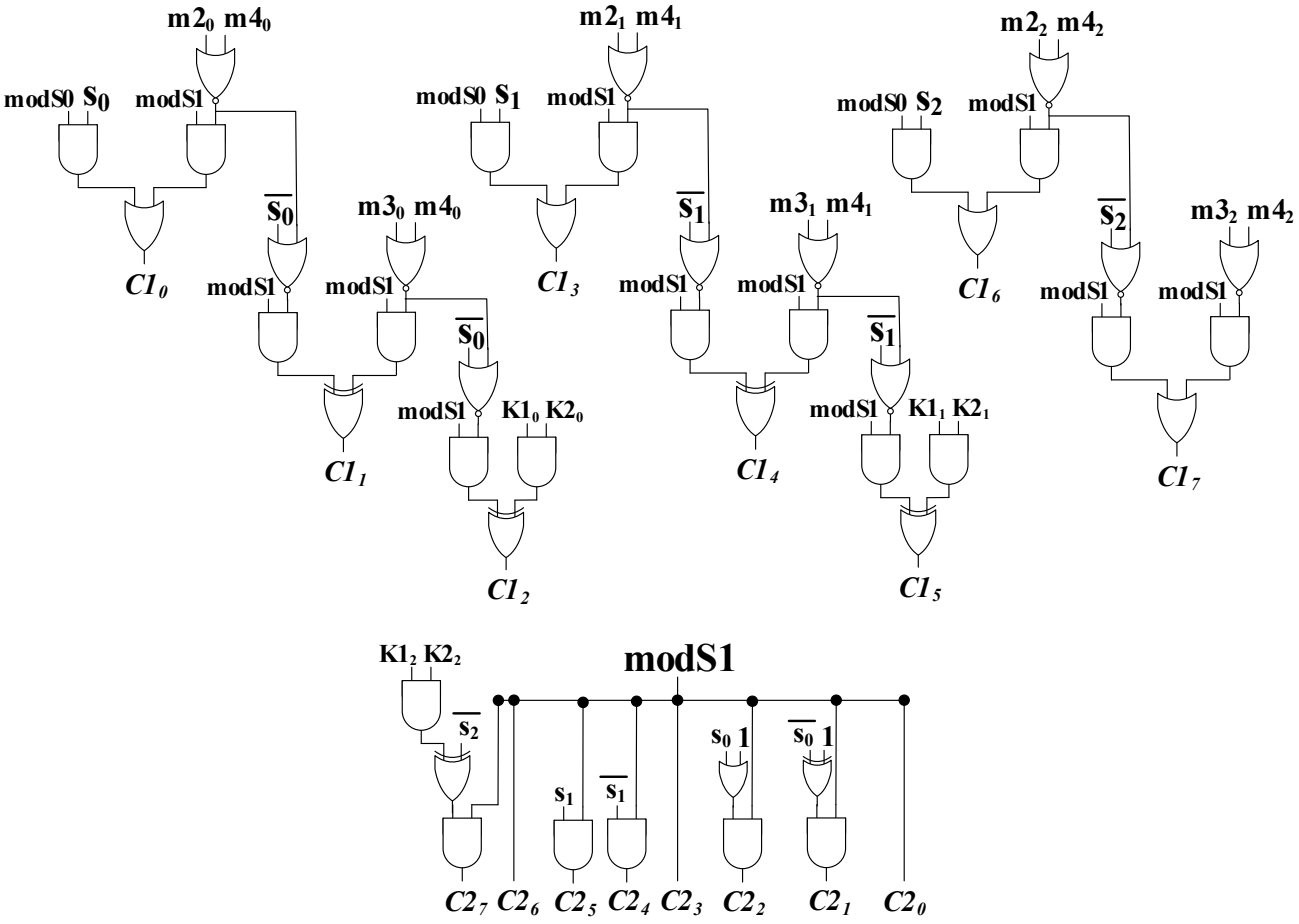

Figure 12. The compensation circuit of the proposed radix-8 multi-modulus multiplier for 8 bits.

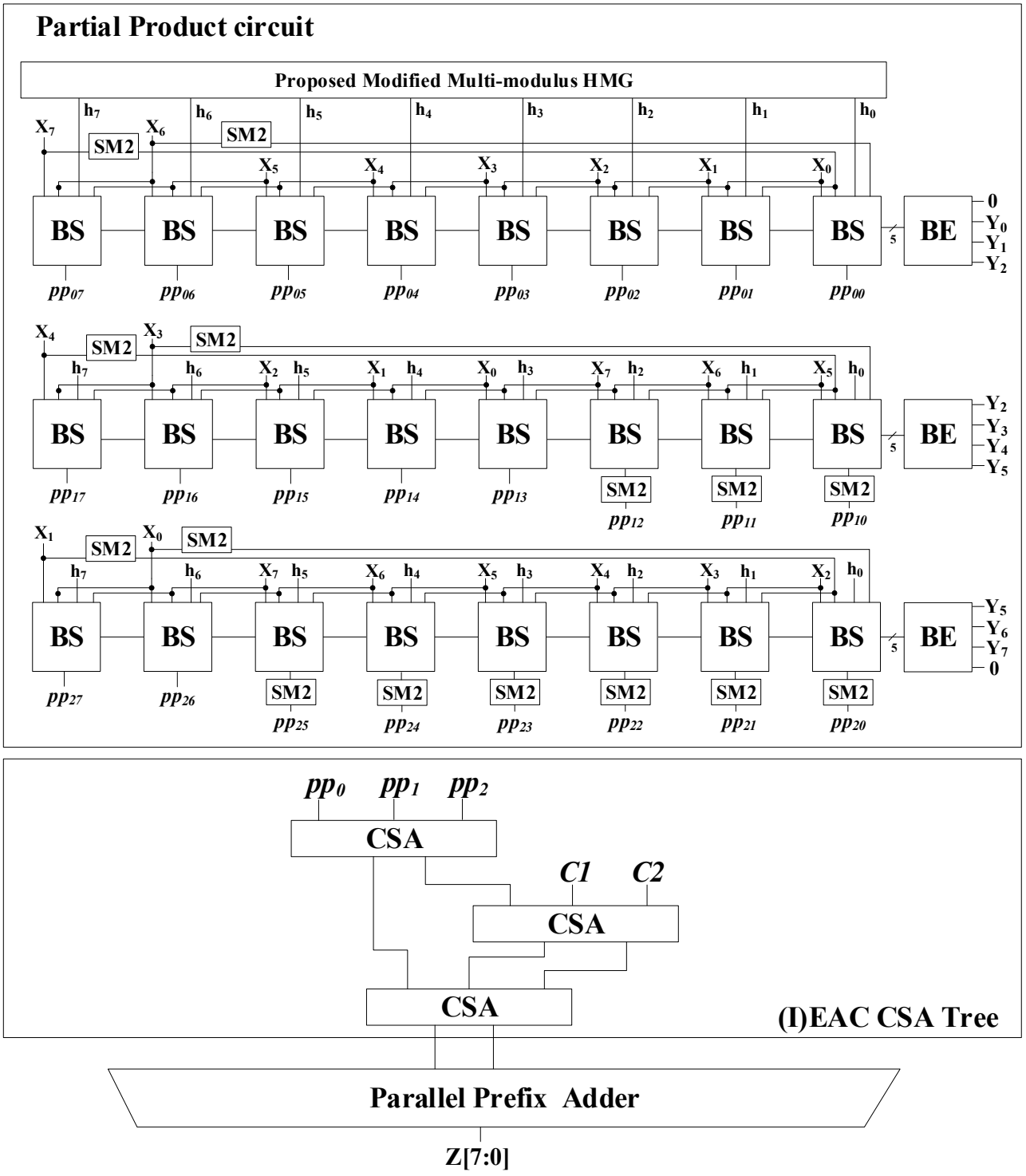

Figure 13. Proposed structure of the radix-8 multi-modulus multiplier for 8 bits (n = 8).

Taking n = 8 as an example for the proposed multi-modulus multiplier based on radix-8 Booth encoding, Figure 15 shows the operational processes of the proposed modulo ($2^n - 1$), modulo (2^n), and modulo ($2^n + 1$) multipliers. For n = 8, for the modulo ($2^n - 1$) multiplication operation with (S1, S0) = (0, 0), A = 141, and B = 221, the final result is 51; for the modulo (2^n) multiplication operation with (S1, S0) = (0, 1), A = 141, and B = 221, the final result is 185; and for the modulo (2^n +1) multiplication operation with (S1, S0) = (1, 0), A = 141, and B = 221, the final result is 64.

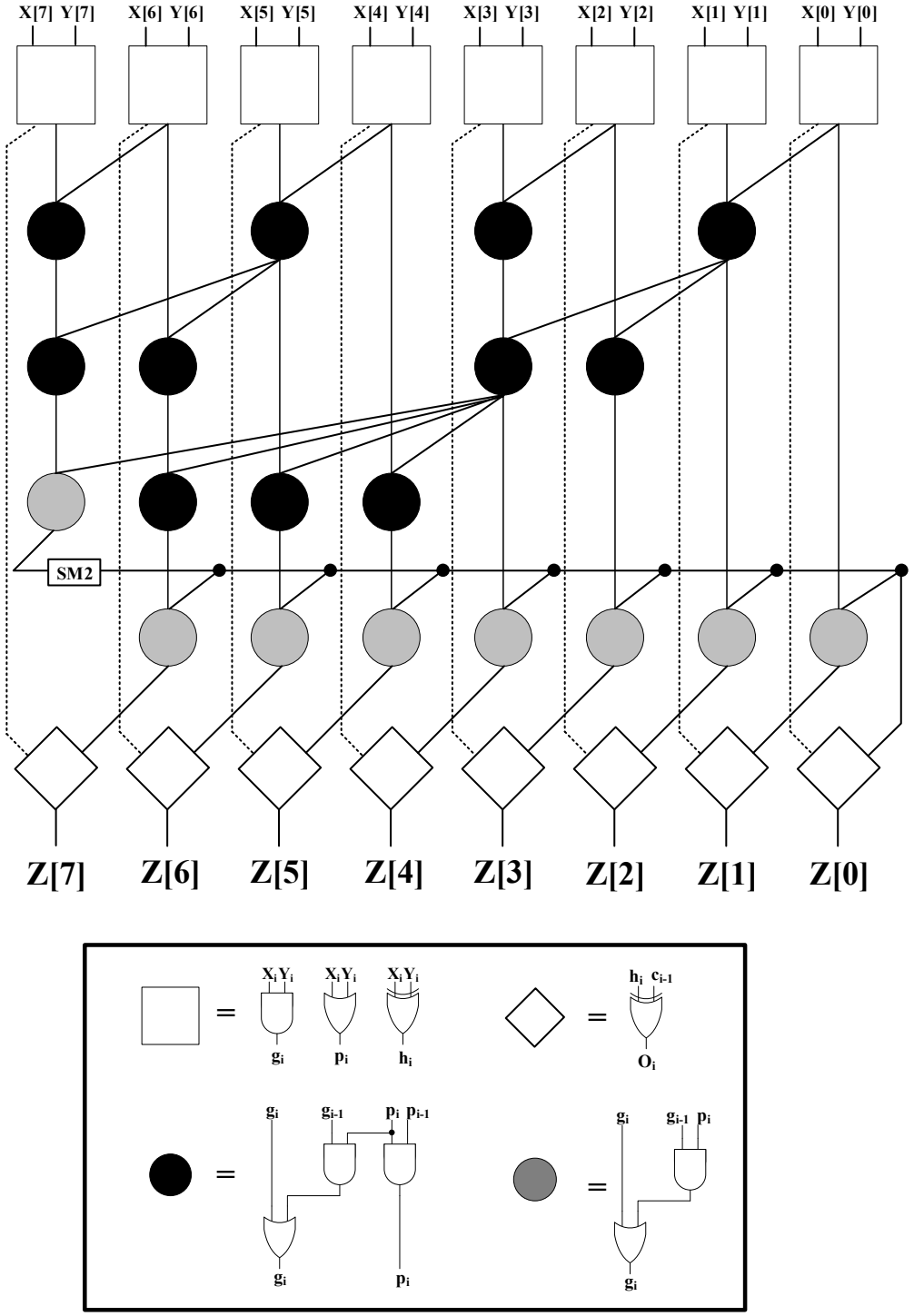

Figure 14. Proposed improved parallel prefix adder of the radix-8 multi-modulus multiplier for 8 bits ($n = 8$).

To summarize, this section presents the design for the multi-modulus HMG and proposed a radix-8 Booth-encoding-based multi-modulus multiplier. The experimental results and comparisons of the hardware area, delay time, dynamic power, area-delay product (ADP), and power-delay product (PDP) with other methods reported in the literature are presented in the next section.

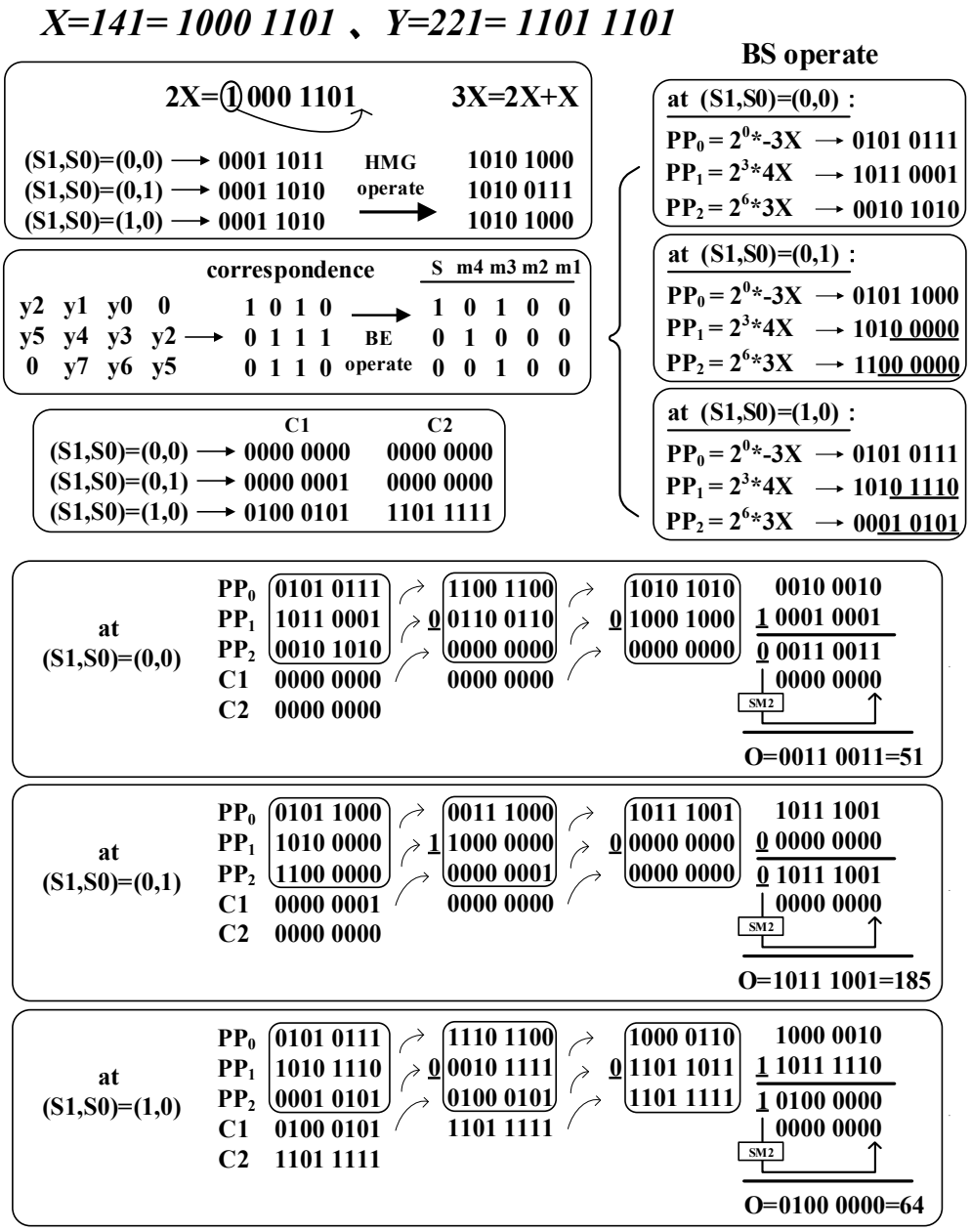

Figure 15. Example of the operational process of the proposed multi-modulus multiplier for 8 bits (n = 8).

4. Experimental Results and Comparison

The proposed structure of the multi-modulus HMG and multi-modulus multipliers based on radix-8 Booth encoding, which is covered in Section 3, is discussed in this section, along with the experimental results and comparison. The proposed multi-modulus HMG structure integrates and improves the HMG used by the modulo ($2^n - 1$) multiplier [22], modulo (2^n) multiplier [23], and modulo ($2^n + 1$) multiplier [24] proposed in the reported studies. The area-saving multifunction based on these three moduli is proposed, and it shares the same hardware architecture. The proposed modified multi-modulus HMG can save 34.48–55.23% of hardware area compared with the reported work [20], as shown in Table 2.

Table 2. Comparison of area of the proposed modified HMG with Muralidharan and Chang [20].

	Muralidharan and Chang [20]	Proposed Modified HMG	
n	**Area (LUT)**	**Area (LUT)**	**Area Saving**
8	29	19	34.48%
16	101	46	54.46%
24	174	96	44.83%
32	267	136	49.06%
40	373	167	55.23%
48	484	230	52.48%

The proposed multi-modulus modulo $(2^n - 1)$, modulo (2^n), and modulo $(2^n + 1)$ multiplexers can support the aforementioned modular multiplication functions in the same circuit hardware. By integrating the individual functions of the modulo $(2^n - 1)$, modulo (2^n), and modulo $(2^n + 1)$ multiplexers into a single multi-modulus multiplier, the proposed approach can save 22.78–35.46% of hardware area compared with previous work [20], as tabulated in Table 3. In addition, the proposed approach can reduce delay time by 4.12–11.15% compared with previous work [20], as tabulated in Table 4. The dynamic power consumption can be reduced by 12.59–24.73% of dissipation power compared with previous work [20], as tabulated in Table 5. Moreover, it can save 27.88–38.88% of ADP compared with previous work [20], as shown in Table 6. Finally, it can save 20.49–27.85% of PDP compared with previous work [20], as tabulated in Table 7.

Table 3. Comparison of area of the proposed multiplier with Muralidharan and Chang [20].

	Muralidharan and Chang [20]	This Work	
n	**Area (LUT)**	**Area (LUT)**	**Area Saving**
8	197	133	32.5%
16	597	461	22.78%
24	1461	943	35.46%
32	2190	1491	31.92%
40	3481	2621	24.71%
48	4970	3560	28.37%

Table 4. Comparison of delay of the proposed multiplier with Muralidharan and Chang [20].

	Muralidharan and Chang [20]	This Work	
n	**Delay (ns)**	**Delay (ns)**	**Delay Saving**
8	19.488	17.37	10.87%
16	25.047	22.254	11.15%
24	31.024	29.74	4.14%
32	33.583	32.166	4.22%
40	39.271	37.614	4.22%
48	39.723	38.086	4.12%

Table 5. Comparison of dynamic power of the proposed multiplier with Muralidharan and Chang [20].

	Muralidharan and Chang [20]	This Work	
n	**Power (W)**	**Power (W)**	**Power Saving**
8	0.054	0.047	13%
16	0.135	0.118	12.59%
24	0.279	0.21	24.73%
32	0.406	0.318	21.67%
40	0.565	0.469	17%
48	0.735	0.584	20.54%

Table 6. Comparison of area-delay product of this work with Muralidharan and Chang [20].

	Muralidharan and Chang [20]			This Work			ADP Saving
n	**Delay (ns)**	**Area (LUT)**	**ADP**	**Delay (ns)**	**Area (LUT)**	**ADP**	
8	19.488	197	3780.04	17.37	133	2310.21	38.88%
16	25.047	597	14,953.06	22.254	461	10,259.09	31.39%
24	31.024	1461	45,326.06	29.74	943	28,044.82	38.13%
32	33.583	2190	73,546.77	32.166	1491	47,959.51	34.80%
40	39.271	3481	136,702.35	37.614	2621	98,586.29	27.88%
48	39.723	4970	197,423.31	38.086	3560	135,586.16	31.32%

Table 7. Comparison of power-delay product of this work with Muralidharan and Chang [20].

	Muralidharan and Chang [20]			This Work			PDP Saving
n	**Delay (ns)**	**Power (W)**	**PDP**	**Delay (ns)**	**Power (W)**	**PDP**	
8	19.488	0.054	1.0524	17.37	0.047	0.8164	22.42%
16	25.047	0.135	3.3813	22.254	0.118	2.6260	22.34%
24	31.024	0.279	8.6557	29.74	0.21	6.2454	27.85%
32	33.583	0.406	13.6347	32.166	0.318	10.2288	24.98%
40	39.271	0.565	22.1881	37.614	0.469	17.6410	20.49%
48	39.723	0.735	29.1964	38.086	0.584	22.2422	23.82%

In Table 2 to Table 7, it is clear that the proposed multi-modulus multiplier based on radix-8 Booth encoding achieves better performance with a lower power, faster operation, greater area-efficiency, and lower ADP and PDP compared with a similar method reported in the literature [20]. The system structure of the proposed approach is compared with that of Muralidharan and Chang [20] in Table 8, showing the weighted system structures adopted for all the modulo multipliers. There are several methods of implementing a multiplier in FPFAs. It can be performed by using LUT, built-in multipliers, internal memory block, and DSP blocks. The LUT method is used in the proposed work. Xilinx field programmable gate array (FPGA) Vivado 2019.2 tools and Verilog hardware description language were used for synthesis and implementation. The Xilinx Artix-7 XC7A35T-CSG324-1 chipset was adopted to evaluate the performance.

Table 8. Comparison of system structure of the proposed multiplier with the work of Muralidharan and Chang [20].

	Item	System Structure
Muralidharan and Chang [20]	Modulo $2^n - 1$	Weighted
	Modulo 2^n	Weighted
	Modulo $2^n + 1$	Diminished-1
This work	Modulo $2^n - 1$	Weighted
	Modulo 2^n	Weighted
	Modulo $2^n + 1$	Weighted

5. Conclusions

A radix-8 weighted Booth-encoded multi-modulus multiplier based on an area-saving hard multiple generator (HMG) is proposed in this paper. Compared with the methods previously reported in the literature, the proposed work can achieve better performance with a circuit design that has a lower power, a faster operation, area-saving, and a lower area-delay product (ADP) and power-delay product (PDP). With the multi-modulus HMG, the proposed architecture can save up to 55.23% (n = 40) of hardware area. With the multi-modulus multiplier, the proposed architecture can save up to 35.46% (n = 24) of hardware area, up to 11.15% (n = 16) of delay time, up to 24.73% (n = 24) of dissipation power, up to 38.88% (n = 8) of ADP, and up to 27.85% (n = 24) of PDP compared with previously reported approaches. The Xilinx field programmable gate array Artix-7 XC7A35T-CSG324-1 chipset was used for synthesis and implementation. The proposed approach can be applied in cryptography, error correction codes, digital signal processors, and other fields.

Author Contributions: Conceptualization, C.-T.K. and Y.-C.W.; Methodology, C.-T.K. and Y.-C.W.; Software, Y.-C.W.; Validation, C.-T.K.; Formal analysis, C.-T.K.; Investigation, C.-T.K. and Y.-C.W.; Writing—original draft, C.-T.K.; Writing—review & editing, C.-T.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ma, S.; Hu, S.; Yang, Z.; Wang, X.; Liu, M.; Hu, J. High Precision Multiplier for RNS $\{2^n - 1, 2^n, 2^n + 1\}$. *Electronics* **2021**, *10*, 1113. [CrossRef]
2. Schoinianakis, D. Residue arithmetic systems in cryptography: A survey on modern security applications. *J. Cryptogr. Eng.* **2020**, *10*, 249–267. [CrossRef]
3. Ramirez, J.; Garcia, A.; Lopez-Buedo, S.; Lloris, A. RNS-enabled Digital Signal Processor Design. *Electron. Lett.* **2002**, *38*, 266–268. [CrossRef]
4. Kalmykov, I.A.; Pashintsev, V.P.; Tyncherov, K.T.; Olenev, A.A.; Chistousov, N.K. Error-Correction Coding Using Polynomial Residue Number System. *Appl. Sci.* **2022**, *12*, 3365. [CrossRef]
5. Juang, T.-B.; Huang, J.-H. Multifunction RNS modulo $(2^n \pm 1)$ Multipliers Based on Modified Booth Encoding. In Proceedings of the 2012 IEEE Asia Pacific Conference on Circuits and Systems, Kaohsiung, Taiwan, 2–5 December 2012; pp. 515–518.
6. Prediger, V.; Bairros, F.; Seman, L.O.; Bezerra, E.A.; Pettenghi, H. RNS processor using moduli sets of the form $2^n \pm 1$. *Int. J. Circuit Theory Appl.* **2023**, *51*, 3432–3442. [CrossRef]
7. Palutla, K.; Gundabathina, P. Implementation of High Speed Modulo $(2^n + 1)$ Multiplier for IDEA Cipher. *Procedia Comput. Sci.* **2020**, *171*, 2016–2022. [CrossRef]
8. Babenko, M.; Nazarov, A.; Deryabin, M.; Kucherov, N.; Tchernykh, A.; Hung, N.V.; Avetisyan, A.; Toporkov, V. Multiple Error Correction in Redundant Residue Number Systems: A Modified Modular Projection Method with Maximum Likelihood Decoding. *Appl. Sci.* **2022**, *12*, 463. [CrossRef]
9. Singhal, S.K.; Mohanty, B.K.; Patel, S.K.; Saxena, G. Efficient Diminished-1 Modulo $(2^n + 1)$ Adder Using Parallel Prefix Adder. *J. Circuits Syst. Comput.* **2020**, *29*, 2050186. [CrossRef]

10. Efstathiou, C.; Kouretas, I.; Kitsos, P. On the modulo $2^n + 1$ addition and subtraction for weighted operands. *Microprocess. Microsyst.* **2023**, *11*, 2138–2164.
11. Patel, B.K.; Kanungo, J. Diminished-1 multiplier using modulo $2^n + 1$ adder. *Int. J. Eng. Technol.* **2018**, *7*, 31–35. [CrossRef]
12. Vergos, H.T.; Bakalis, D. Area-time efficient multi-modulus adders and their applications. *Microprocess. Microsyst.* **2012**, *42*, 409–419. [CrossRef]
13. Zimmermann, Z. Efficient VLSI Implementation of Modulo $(2^n \pm 1)$ Addition and Multiplication. In Proceedings of the 14th IEEE Symposium on Computer Arithmetic, Adelaide, Australia, 14–16 April 1999; pp. 158–167.
14. Efstathou, C.; Moshopoulos, N.; Axelos, N.; Pekmestzi, K. Efficient modulo $2^n + 1$ multiply and multiply-add units based on modified Booth encoding. *Integration* **2014**, *47*, 140–147. [CrossRef]
15. Vergos, H.T.; Efstathiou, C. Design of efficient modulo $2^n + 1$ multipliers. *IET Comput. Digit. Tech.* **2007**, *1*, 49–57. [CrossRef]
16. Chen, J.W.; Yao, R.H.; Wu, W.J. Efficient modulo $2^n + 1$ multipliers. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2011**, *19*, 2149–2157. [CrossRef]
17. Sousa, L.; Chaves, R. A universal architecture for designing efficient modulo $2^n + 1$ multipliers. *IEEE Trans. Circuits Syst. I* **2005**, *52*, 1166–1178. [CrossRef]
18. Juang, T.-B.; Kuo, C.-T.; Wu, G.-L.; Huang, J.-H. Multifuction RNS Modulo $2^n \pm 1$ Multipliers. *J. Circuits Syst. Comput.* **2012**, *21*, 1250027. [CrossRef]
19. Muralidharan, R.; Chang, C.-H. Area-Power Efficient Modulo $2^n - 1$ and Modulo $2^n + 1$ Multipliers for $\{2^n - 1, 2^n, 2^n + 1\}$ Based RNS. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2012**, *59*, 2263–2274. [CrossRef]
20. Muralidharan, R.; Chang, C.-H. Radix-4 and Radix-8 Booth Encoded Multi-Modulus Multipliers. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2013**, *60*, 2940–2952. [CrossRef]
21. Kumar, R.; Jaiswal, R.K.; Mishra, R.A. Perspective and Opportunities of Modulo $2^n - 1$ Multipliers in Residue Number System: A Review. *J. Circuits Syst. Comput.* **2020**, *29*, 2030008. [CrossRef]
22. Kabra, N.K.; Patel, Z.M. Area and power efficient hard multiple generator for radix-8 modulo $2^n - 1$. *Integr. VLSI J.* **2020**, *75*, 102–113. [CrossRef]
23. Kabra, N.K.; Patel, Z.M. A radix-8 modulo 2^n multiplier using area and power-optimized. *IET Comput. Digit. Tech.* **2021**, *15*, 36–55. [CrossRef]
24. Mirhosseini, S.M.; Molahosseini, A.S. A Reduced-Bias Approach with a Lightweight Hard-Multiple Generator to Design Radix-8 Modulo $2^n + 1$ Multiplier. *IEEE Trans. Circuits Syst. II Express Briefs* **2017**, *64*, 817–821.
25. Kuo, C.-T.; Wu, Y.-C. FPGA Implementation of a Novel Multifunction Modulo $(2^n \pm 1)$ Multiplier Using Radix-4 Booth Encoding Scheme. *Appl. Sci.* **2023**, *13*, 10407. [CrossRef]
26. Fu, C.; Zhu, X.; Huang, K.; Gu, Z. An 8-bit Radix-4 non-volitile parallel multiplier. *Electronics* **2021**, *10*, 2358. [CrossRef]