Depth Optimized Synthesis of Symmetric Boolean Functions

Martha Schnieber Saman Froehlich Rolf Drechsler
Cyber-Physical Systems, DFKI GmbH and Group of Computer Architecture, University of Bremen, Germany
Martha.Schnieber@dfki.de froehlich@uni-bremen.de drechsler@uni-bremen.de

***Abstract*—Symmetric Boolean functions are characterized by the fact that their output can be determined by the Hamming weight of their inputs. Symmetric Boolean functions are essential for many complex systems and possess significant cryptographic properties.**

Due to their popularity and significance, research has been focusing on optimization of synthesis strategies for symmetric Boolean functions. However, none of them has targeted optimizing the depth of the final synthesis, yet.

In this paper, we propose a synthesis scheme for symmetric Boolean functions which generates circuits with a worst-case depth of $O(\log^2 n)$. To the best of our knowledge we are the first to propose a synthesis scheme that aims at reducing the depth of the generated circuits for symmetric Boolean functions. In the experiments we show that our approach allows to reduce the depth of the final implementation by up to 25.93% compared to the state-of-the-art.

I. Introduction

Symmetric Boolean functions are characterized by the fact that their output can be determined by the Hamming weight of their inputs. Research has focused on generating symmetric Boolean functions which are correlation immune [1], nonlinear [2], [3] and possess algebraic immunity [4], [5] and thus include all cryptographic properties presented in [6] and are of cryptographic significance. Due to their significant cryptographic properties, symmetric Boolean functions are essential for many complex systems [7] (e.g., stream ciphers, block ciphers, or hash functions [8]).

As symmetric Boolean functions can be represented in very compact ways, the required memory space and the complexity of the hardware implementation are very limited. Specially, it has been shown that all symmetric Boolean functions can be computed by a circuit with a depth of $O(\log n)$ in theory [9], however, no methodology which results in such a circuit has been proposed, yet.

Due to their popularity and significance, research has been focusing on optimization of synthesis strategies for symmetric Boolean functions. These strategies often consist of three steps (see Figure 1): (1) Synthesis of special symmetric Boolean functions (such as unate symmetric Boolean functions or Matriochka symmetric Boolean functions), (2) composing them to elementary symmetric Boolean functions, (3) combining the elementary symmetric functions to synthesize arbitrary symmetric Boolean functions. Often the first step is the most complex one and consists of multiple stages. While the focus has been on reducing the complexity of the final implementation in terms of area (e.g. [10], [11], [12]), none of them has targeted optimizing the depth of the final synthesis, yet and they all have a worst-case complexity of $O(n)$ at best.

This work was supported by the German Research Foundation (DFG) within the Project *PLiM* (DR 287/35-1) and the Reinhart Koselleck Project *PolyVer* (DR 287/36-1).

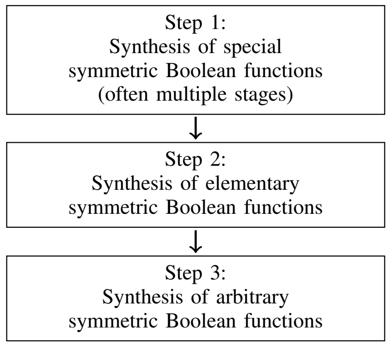


Fig. 1. General structure of synthesis strategies

In this paper, we are the first to propose a synthesis scheme for symmetric Boolean functions that generates circuits with below linear depth scaling. Our proposed synthesis scheme is applicable to any symmetric Boolean function and results in circuits with a worst-case depth of $O(\log^2 n)$. We propose to use a three-step approach, where a novel third stage in the first step and a novel third step result in significant depth reductions. Finally, we provide a proof that we can synthesize arbitrary symmetric Boolean functions with a worst-case depth of $O(\log^2 n)$ using our proposed synthesis scheme. In the experiments, we show that our approach allows to reduce the depth by up to 25.93% compared to the state-of-the-art.

The remainder of this paper is structured as follows: In Section II, related work is presented. Section III introduces preliminary knowledge. Our proposed synthesis scheme for symmetric Boolean functions is presented in Section IV. In Section V, our proposed synthesis scheme is evaluated experimentally and compared to the state-of-the-art. Finally, Section VI concludes the paper.

II. Related Work

In this section, related work is reviewed and the differences to our synthesis scheme are shown. As synthesis schemes for symmetric Boolean functions often consist of the three steps shown in Figure 1, we refer to these steps. We also refer to the stages the first step often consists of.

The authors of [10] present a scheme for symmetric Boolean functions. In the first step, a cellular network is used as first stage for the synthesis of unate symmetric functions, which are decomposed in the next stage. As final, third stage a cascade of cells is introduced. Specially, this last stage has a worst-case delay of $O(n)$. As the second stage is a recursion of all three stages, this results in a worst-case delay of $O(n \cdot \log n)$. Due to the optimization of the third stage in the first step and the optimized third step, our proposed methodology only has a delay of $O(\log^2 n)$.

The work [11] presents a synthesis method for symmetric Boolean functions with focus on generating networks with complete and robust path testability and is similar to that of [10]. Even though optimizations have been proposed to guarantee testability, the complexity remains $O(n \cdot \log n)$.

In [12], the authors propose a three-step network for the synthesis of symmetric Boolean functions with focus on area reduction. The first step consists of a half-adder network which implements Matriochka Symmetric functions. This first step has the depth $O(n)$. In the second step, an inverter-AND network is used and an OR-network in the third stage. In contrast, our synthesis scheme uses an AND-OR network in the first step and in the third step. While the third step is also an OR-network, our third step has only logarithmic depth and the overall depth of our synthesis scheme scales better. Further, the authors of [12] do not provide any benchmarks on arbitrary symmetric Boolean functions, but only on Matriochka Symmetric functions.

III. Preliminaries

In this section, we introduce basic knowledge about symmetric Boolean functions. First, we introduce symmetric Boolean functions in Section III-A. Subsequently, consecutive symmetric functions and their relation to arbitrary symmetric Boolean functions are presented in III-B. Finally, we introduce unate symmetric Boolean functions and show how to generate consecutive symmetric functions from unate symmetric functions in Section III-C.

A. Symmetric Boolean Functions

A function $f(x_1, \ldots, x_n)$ with n inputs and one output is symmetric, if f is invariant under all possible permutations of the inputs, meaning the output only depends on the number of inputs that are set to true. We denote a symmetric function by $S^n(A)$ with $A \subseteq \{0, 1, \ldots, n\}$, which has n inputs and evaluates to true, if and only if exactly A_m inputs are set to true for one $A_m \in A$ (see [9]). E.g., $S^{10}(2, 6, 8)$ evaluates to true if and only if either exactly 2, 6 or 8 of the 10 inputs are set to true.

B. Consecutive Symmetric Boolean Functions

A symmetric function $S^n(A)$ with n inputs and one output is consecutive if $A_{m+1} = A_m + 1$ for every $A_m, A_{m+1} \in A$, meaning A consists of consecutive numbers, and is denoted as $S^n(A_0 - A_j)$, where A_0 is the first number of A and A_j is the last number of A. Thus, $S^{10}(3-7) = S^{10}(3, 4, 5, 6, 7)$ is a consecutive symmetric function. Each symmetric function can be divided into several consecutive symmetric functions:

$$S^n(A) = S^n(A_1) \vee S^n(A_1) \vee \cdots \vee S^n(A_i)$$

E.g. $S^{16}(1, 2, 3, 6, 9, 10, 15) = S^{16}(1-3) \vee S^{16}(6) \vee S^{16}(9-10) \vee S^{16}(15)$ [10].

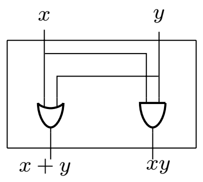


Fig. 2. AND-OR cell

C. Unate Symmetric Boolean Functions

A positive unate symmetric function $u_i(n)$ with n inputs and one output is equivalent to $S^n(i-n)$, meaning the function evaluates to true if and only if at least i of the n input variables are set to true. A negative unate symmetric function evaluates to true if at most i inputs are set to true. However, in this paper, we will use exclusively positive unate functions and will therefore refer to them as unate functions. Each unate symmetric function is also consecutive and furthermore, each consecutive symmetric function can be expressed with unate symmetric functions:

$$\begin{aligned} S^n(A_0 - A_j) &= S^n(A_0 - n) \wedge \neg S^n(A_j + 1 - n) \\ &= u_{A_0}(n) \wedge \neg u_{A_j+1}(n) \end{aligned}$$

For example, $S^{16}(5-9) = u_5(16) \wedge \neg u_{10}(16)$ (see [11]). In this paper, we substitute $u_i(n)$ with u_i if the corresponding n is evident from the context.

IV. Synthesis of Symmetric Boolean Functions

In this section, we present our novel synthesis scheme. First, we describe how to synthesize unate symmetric functions. Based on this, we show how consecutive symmetric functions and general non-consecutive symmetric functions can be realized. Our synthesis scheme is therefore split into three steps. Furthermore, we will provide an analysis of the depth for each step.

Step 1: Synthesis of Unate Symmetric Functions

In the first step, we construct Module(n), which has n inputs and n outputs, where the i-th output realizes the unate symmetric function u_i. We propose an improvement to the synthesis of unate symmetric functions in [11], such that the worst-case depth is constrained to $O(\log^2 n)$. The synthesis uses AND-OR cells, which are shown in Figure 2.

We define Module(n) for $n = 2^k$. For $2^{k-1} < n < 2^k$, Module(n) can be constructed by building Module(2^k), setting all inputs x_i with $i > n$ to 0 and optimizing the resulting circuit, i.e. removing all affected gates. Thus, for the remainder of this section, we focus on the case $n = 2^k$.

Module(n) consists of three stages:

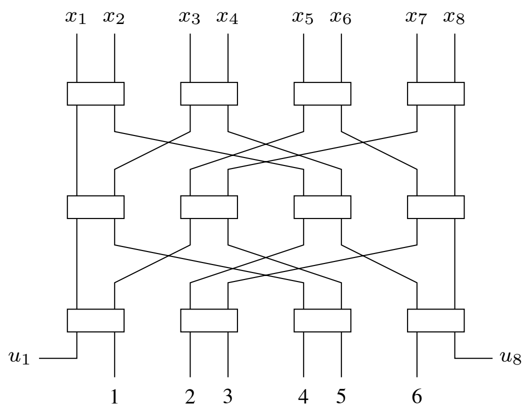


Fig. 3. First Stage of Module(8)

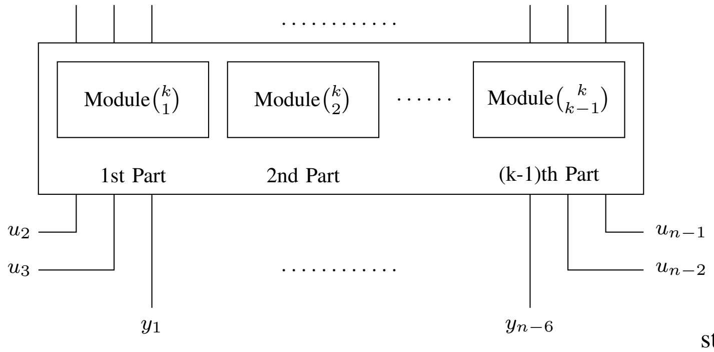


Fig. 4. Second Stage

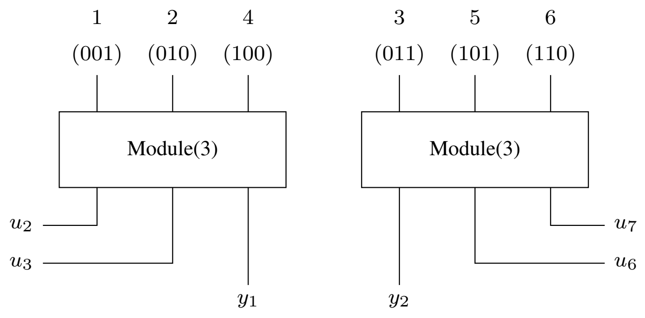
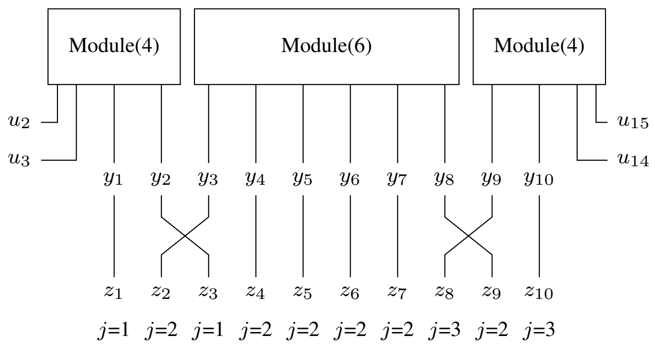


Fig. 5. Second Stage for $n = 8$

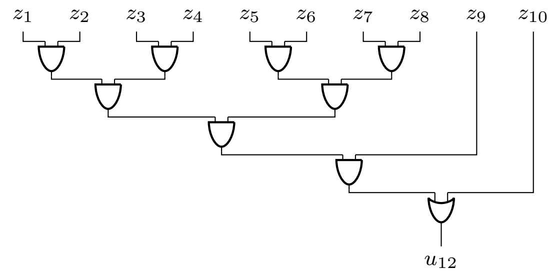
1) First Stage: The first stage consists of $\log n$ layers, where each layer is built of $\frac{n}{2}$ AND-OR cells. For Module(8), the first stage is shown in Figure 3 where each box corresponds to one AND-OR cell. The layers are connected as in shuffle-exchange networks [13]. The first stage has n outputs, where the first output realizes the final unate symmetric function u_1 and the last output realizes the final symmetric function u_n, because the first output is an OR-tree of all inputs and the last output is an AND-tree of all inputs. All outputs but u_1 and u_n are passed on to the second stage.

2) Second Stage: The second stage has $n-2$ inputs, which are divided into $k-1$ parts. The inputs of the second stage are numbered from 1 to $n-2$. The j-th part of the second



Module(4)
Module(6)
Module(4)
u_2
u_3
u_{15}
u_{14}
y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9 y_{10}
z_1 z_2 z_3 z_4 z_5 z_6 z_7 z_8 z_9 z_{10}
j=1 j=2 j=1 j=2 j=2 j=2 j=2 j=3 j=2 j=3

Fig. 6. Connection from second stage to third stage for $n = 16$



z_1 z_2 z_3 z_4 z_5 z_6 z_7 z_8 z_9 z_{10}
u_{12}

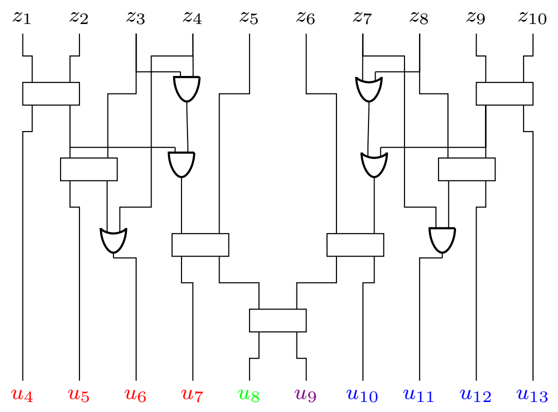
Fig. 7. Third stage for u_{12}

stage consists of all inputs where the binary representation of the corresponding number has j ones. Thus, the second stage consists of $k-1$ parts, where each j-th part is Module$\binom{k}{j}$, i.e. a recursive call of Module(m), as shown in Figure 4. An example of the second stage is shown in Figure 5 for $n = 8$. For $n = 8$, the second stage has $n-2 = 6$ inputs and two recursive calls of Module(3). The first Module(3) has the inputs 1, 2 and 4 with the binary representations (001), (010) and (100) since they each have exactly one one in their binary representation. The second Module(3) has the inputs 3, 5 and 6 with the binary representations (011), (101) and (110) since they each have exactly two ones in their binary representation.

If the i-th output of the j-th part of the second stage evaluates to true, at least i conjunctions of 2^j inputs of the first stage are all true. For example, if for Module(8) in Figure 5 the third output of the first part (i.e., the left Module(3)) evaluates to true, there are at least three combinations of 2 of the 8 inputs, which evaluate to true.

The second stage has $n-2$ outputs. If the first output of the second stage evaluates to true, at least 2 of the inputs of the first stage are true, etc. Therefore, the first two outputs realize u_2 und u_3, whereas the last two outputs realize u_{n-2} and u_{n-1}.

3) Third Stage: The third stage has $n-6$ inputs $y_1, y_2 \dots y_{n-6}$. First, these inputs are rearranged. We call these rearranged inputs $z_1, z_2 \dots z_{n-6}$, which are assigned sequentially from z_1 to z_{n-6}. For $i \leq \frac{n-6}{2}$, we set j as the number of ones in the binary representation of $2(i+1)$ and z_i is the first output of the j-th part of the second stage which is neither u_2, u_3, u_{n-2}, u_{n-1}, or already assigned to another



z_1 z_2 z_3 z_4 z_5 z_6 z_7 z_8 z_9 z_{10}
u_4 u_5 u_6 u_7 u_8 u_9 u_{10} u_{11} u_{12} u_{13}

Fig. 8. Third stage for $n = 16$

z_a with $a < i$. For $i > \frac{n-6}{2}$ the process is similar, except that we set j as the number of ones in the binary representation of $2(i-5)+1$. Figure 6 shows these connections for $n = 16$. At the top, the outputs of the second stage are shown, whereas on the bottom, the rearranged inputs of the third stage are shown, as well as the corresponding value of j.

In [11], the third stage consists of a cascade of cells, which leads to a linear depth. Here, we present a novel third stage which has a logarithmic depth.

Each output u_{i+3} of this stage can be described using a distinction of cases.

1) If $i < \frac{n-6}{2}$, u_{i+3} is the disjunction of the input z_{i+1} and the conjunction of all inputs z_j with $j \leq i$, i.e. $u_{i+3} = z_{i+1} \vee (z_1 \wedge z_2 \wedge \cdots \wedge z_i)$.
2) If $i > \frac{n-6}{2} + 1$, u_{i+3} is the conjunction of the input z_{i-1} and the disjunction of all inputs z_j with $j \geq i$, i.e. $u_{i+3} = z_{i-1} \wedge (z_i \wedge z_{i+1} \wedge \cdots \wedge z_{n-6})$.
3) If $i = \frac{n-6}{2}$, $u_{i+3} = (z_1 \wedge z_2 \wedge \cdots \wedge z_i) \vee (z_{i+1} \vee z_{i+1} \vee \cdots \vee z_{n-6})$.
4) If $i = \frac{n-6}{2} + 1$, $u_{i+3} = (z_1 \wedge z_2 \wedge \cdots \wedge z_{i-1}) \wedge (z_i \vee z_{i+1} \vee \cdots \vee z_{n-6})$.

Thus, we propose to realize each output u_{i+3} of the third stage depending on i and the described cases:

1) with a logarithmic tree of conjunctions followed by a disjunction if $i < \frac{n-6}{2}$,
2) with a logarithmic tree of disjunctions followed by a conjunction if $i > \frac{n-6}{2} + 1$,
3) with a disjunction of two logarithmic trees if $i = \frac{n-6}{2}$ or
4) with a conjunction of two logarithmic trees if $i = \frac{n-6}{2} + 1$.

In Figure 7, an example for u_{12} is shown for the case that $9 \leq \frac{n-6}{2}$. As described, u_{12} is calculated by a logarithmic tree of conjunctions, i.e. a tree of AND-gates, followed by a disjunction, i.e. an OR-gate. The complete third stage for $n = 16$ is shown in Figure 8, where the outputs for case 1) are marked in red (u_4, u_5, u_6, u_7), case 2) in blue (u_{10}, u_{11}, u_{12}, u_{13}), case 3) in green (u_8) and case 4) in violet (u_9).

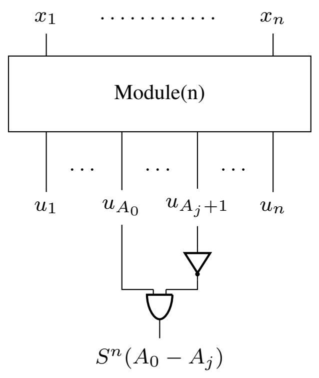
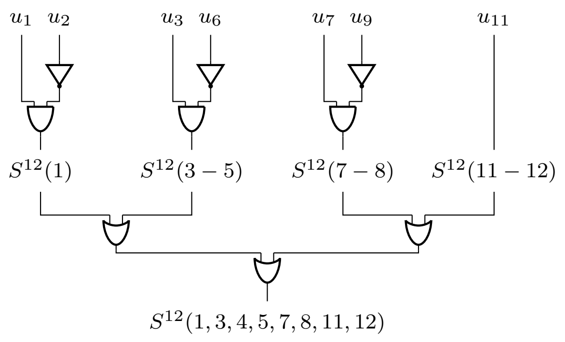


Fig. 9. Consecutive Symmetric Function $S^n(A_0 - A_j)$



u_1 u_2 u_3 u_6 u_7 u_9 u_{11}
$S^{12}(1)$
$S^{12}(3-5)$
$S^{12}(7-8)$
$S^{12}(11-12)$
$S^{12}(1,3,4,5,7,8,11,12)$

Fig. 10. Non-consecutive Symmetric Function $S^{12}(1, 3, 4, 5, 7, 8, 11, 12)$

Step 2: Synthesis of Consecutive Symmetric Functions

Every consecutive and not unate symmetric function can be realized with two unate symmetric functions, since $S^n(A_0 - A_j) = u_{A_0}(n) \wedge \neg u_{A_j+1}(n)$. Thus, to realize $S^n(A_0 - A_j)$, an AND-gate and a NOT-gate have to be added to Module(n) as in [11], which is shown in Figure 9.

Step 3: Synthesis of Non-consecutive Symmetric Functions

As every non-consecutive symmetric function can be divided into at most $\frac{n}{2}$ consecutive symmetric functions, every non-consecutive symmetric function can be synthesized by a disjunction of the outputs of consecutive symmetric functions. To maintain a logarithmic depth, we propose to realize the disjunction as a logarithmic tree. An example of this is shown in Figure 10 for $S^{12}(1, 3, 4, 5, 7, 8, 11, 12)$.

Depth Analysis

Theorem 1. *The first stage of Step 1 has a depth of* $\lceil \log n \rceil$.

Proof. The first stage consists of $\lceil \log n \rceil$ consecutive levels and there is no connection from level i to level j for every $j \leq i$. Thus, the depth is $\lceil \log n \rceil$. □

We analyze the second stage after the third stage, as the second stage consists of recursive calls of all three stages of Module.

Theorem 2. *The third stage of Step 1 has a depth of* $\lceil \log((n - 6)/2) \rceil + 1$.

Proof. The ouputs with the largest depth are $z_{(n-6)/2}$ and $z_{(n-6)/2+1}$. We realize these as a conjunction or disjunction of two logarithmic trees. Both of these logarithmic trees have a depth of $\lceil \log((n-6)/2) \rceil$, as each tree has $(n-6)/2$ inputs. Therefore, after the conjunction or disjunction, the depth of this stage is $\lceil \log((n-6)/2) \rceil + 1$. $\square$

Theorem 3. *The depth of the second stage of Step 1 has an upper bound of* $2\lceil \log^2 n \rceil - 3\lceil \log n \rceil + 2$.

Proof. The second stage consists of multiple sub-modules. The largest and deepest sub-module is Module$\binom{\lceil \log n \rceil}{(\lceil \log n \rceil)/2}$. In the second stage of Module$\binom{\lceil \log n \rceil}{(\lceil \log n \rceil)/2}$, there are more sub-modules. However, it holds that $\lceil \log \binom{\lceil \log i \rceil}{(\lceil \log i \rceil)/2} \rceil < \lceil \log i \rceil$ for each i. Thus, the logarithm of the number of inputs is reduced with each recursive call and the number of recursive calls is therefore bounded by $\lceil \log n \rceil$. Additionally, each call of Module(m) has a first and third stage, which have a depth of at most $\lceil \log m \rceil$ and $\lceil \log m \rceil + 1$ respectively, as shown in Theorem 1 and Theorem 2. Thus, the overall depth of the second stage is bounded by:

$$\begin{aligned} D(n) &\leq \sum_{i=2}^{\lceil \log n \rceil - 1} (\lceil \log i \rceil + \lceil \log i \rceil + 1) \\ &= \sum_{i=2}^{\lceil \log n \rceil - 1} (2\lceil \log i \rceil + 1) \\ &\leq (\lceil \log n \rceil - 1) \cdot (2(\lceil \log n \rceil - 1) + 1) \\ &= 2\lceil \log^2 n \rceil - 3\lceil \log n \rceil + 2 \end{aligned}$$

$\square$

Theorem 4. *Step 2 has a depth of* 2.

Proof. Step 2 adds a NOT-gate and a consecutive AND-gate. Thus, the depth is 2. $\square$

Theorem 5. *The depth of Step 3 has an upper bound of* $\lceil \log \frac{n}{2} \rceil$.

Proof. A symmetric function can be split into at most $\frac{n}{2}$ consecutive symmetric functions. Thus, the logarithmic tree of OR-gates has at most $\frac{n}{2}$ inputs and has therefore a depth of at most $\lceil \log \frac{n}{2} \rceil$ $\square$

Theorem 6. *The depth of the whole circuit is* $O(\log^2 n)$

Proof. Connecting all stages and steps, the depth is:

$$\begin{aligned} &O(2\lceil \log^2 n \rceil - 2\lceil \log n \rceil + \lceil \log((n-6)/2) \rceil + \lceil \log \frac{n}{2} \rceil + 5) \\ =&O(\log^2 n) \end{aligned}$$

$\square$

Thus, we have shown, that our synthesis guarantees sublinear depth, in particular a depth of $O(\log^2 n)$.

TABLE I
NUMBER OF GATES AND MAXIMUM DEPTH ON UNATE SYMMETRIC FUNCTIONS FROM $n = 2$ TO $n = 32$

n	Number of Gates		Depth		
	[11]	Proposed	[11]	Proposed	Reduction
2	2	2	1	1	-
3	6	6	3	3	-
4	10	10	3	3	-
5	18	18	7	7	-
6	24	24	7	7	-
7	32	32	7	7	-
8	38	38	7	7	-
9	52	56	12	11	8.33%
10	62	68	12	12	-
11	78	80	12	15	-
12	84	88	13	15	-
13	98	100	13	15	-
14	108	110	13	15	-
15	118	120	13	15	-
16	126	128	13	15	-
17	144	185	24	18	25%
18	166	197	24	19	20.83%
19	170	213	24	21	12.5%
20	186	227	24	21	12.5%
21	206	243	24	22	8.33%
22	220	255	24	22	8.33%
23	236	269	24	22	8.33%
24	248	282	24	22	8.33%
25	264	294	24	22	8.33%
26	278	312	26	22	15.38%
27	300	330	28	23	17.86%
28	314	344	28	23	17.86%
29	332	358	28	23	17.86%
30	344	372	28	23	17.86%
31	360	386	28	23	17.86%
32	370	396	28	23	17.86%

V. Experimental Results

To evaluate our method, we implemented our proposed scheme as well as the approach proposed in [11] for Module(2^k) in Verilog for $k \leq 5$. For Module(n) with $n < 2^k$, we built Module(2^k) and optimized the resulting circuits with Yosys 0.9 [14] for both, our proposed synthesis scheme and the approach proposed in [11]. First, we evaluate the improvement of the depth of Module(n) compared to [11] in Section V-A. Subsequently, we evaluate the depth of consecutive and non-consecutive symmetric Boolean functions in Section V-B.

A. Depth of Module(n)

In Table I, we compare our proposed methodology for the synthesis of Module(n) for $2 \leq n \leq 32$ to that of [11]. In the Columns 2-3, we compare the number of gates needed for our proposed method to that of [11]. Columns 4-5 show the depth of the synthesized circuits and Column 6 shows the reduction in %. As can be seen, the results for $n \leq 8$ are equal, because the third stage for $n \leq 8$ consists of at most one AND-OR

TABLE II
DEPTH ON SYMMETRIC FUNCTIONS

Symmetric Function	[11]	Proposed	Reduction
S^{17}(1-8)	19	18	5.26%
S^{17}(5-7,9,11-12)	24	21	12.5%
S^{20}(1-5,13-14)	26	22	15.38%
S^{20}(7-9,16-19)	26	23	11.54%
S^{22}(13)	24	21	12.5%
S^{22}(6-10,15-22)	24	20	16.67%
S^{25}(9-17)	25	22	12%
S^{25}(1-8,13-16,18-19,22-25)	28	26	7.14%
S^{27}(3-22)	24	24	-
S^{27}(1,15,22)	32	27	15.63%
S^{29}(19)	27	24	11.11%
S^{29}(1-15,21)	26	21	19.23%
S^{32}(13-20,31)	27	20	25.93%
S^{32}(2-12,14-15,18,21-23,25-32)	33	28	15.15%

cell and therefore, our method is equivalent to the method proposed in [11]. For $11 \leq n \leq 16$, the depth of the circuits generated by our proposed methodology is slightly increased. The reason for this is, that for $9 \leq n \leq 16$, the depths of the third stages of both methods are very similar, with the third stage of [11] having a depth of 5, and our third stage having a depth of 4. As the first and last outputs of the second stage have a low depth compared to the other outputs of the second stage, the third stage of [11] can already start in the second stage and is therefore partially shifted into the second stage. Because of the logarithmic tree, our third stage cannot be shifted into the second stage and therefore our results are slightly worse for $11 \leq n \leq 16$. For $n \geq 17$ however, our method shows a depth reduction of up to 25%. As our third stage requires more gates, we report a slight increase in the number of gates compared to [11].

B. Depth of Symmetric Boolean Functions

Table II shows the depth of exemplary consecutive and non-consecutive symmetric Boolean functions using the proposed method and compares the results to [11]. As it is not stated in [11] how the disjunction of step 3 is implemented, we use a logarithmic tree of disjunctions for the results of both our method and the method from [11]. Using our proposed method, we can report a depth reduction of up to 25.93%.

In practice, our proposed method generally reduces the depth in comparison to [11]. Even though the reduction varies, the worst case of [11] can reach a depth of $O(n \log n)$, whereas our method guarantees a depth of $O(\log^2 n)$ in the worst case.

VI. CONCLUSION

In this paper, we have presented a novel scheme for the synthesis of symmetric functions and have shown that our method guarantees a depth of $O(\log^2 n)$. The synthesis scheme consists of three steps, where the first step realizes unate symmetric functions, the second step realizes consecutive functions and the third step realizes non-consecutive symmetric functions. In our experiment, we have been able to report a depth reduction of up to 25% on unate symmetric functions and up to 25.93% on symmetric functions compared to the state-of-the-art.

REFERENCES

[1] P. Sarkar and S. Maitra, "Balancedness and correlation immunity of symmetric boolean functions," *Discrete Math.*, vol. 307, no. 1920, p. 23512358, Sep. 2007. [Online]. Available: https://doi.org/10.1016/j.disc.2006.08.008

[2] C. Carlet, "On the degree, nonlinearity, algebraic thickness, and nonnormality of boolean functions, with developments on symmetric functions," *IEEE Transactions on Information Theory*, vol. 50, no. 9, pp. 2178–2185, 2004.

[3] S. Maitra and P. Sarkar, "Maximum nonlinearity of symmetric boolean functions on odd number of variables," *IEEE Transactions on Information Theory*, vol. 48, no. 9, pp. 2626–2630, 2002.

[4] L. Qu, K. Feng, F. Liu, and L. Wang, "Constructing symmetric boolean functions with maximum algebraic immunity," *IEEE Transactions on Information Theory*, vol. 55, no. 5, pp. 2406–2412, 2009.

[5] Na Li and Wen-Feng Qi, "Symmetric boolean functions depending on an odd number of variables with maximum algebraic immunity," *IEEE Transactions on Information Theory*, vol. 52, no. 5, pp. 2271–2273, 2006.

[6] Y. Xianyang and B. Guo, "Further enumerating boolean functions of cryptographic significance," *J. Cryptol.*, vol. 8, no. 3, p. 115122, Sep. 1995. [Online]. Available: https://doi.org/10.1007/BF00202268

[7] G. Gao, Y. Guo, and Y. Zhao, "Recent results on balanced symmetric boolean functions," *IEEE Transactions on Information Theory*, vol. 62, no. 9, pp. 5199–5203, 2016.

[8] A. Canteaut and M. Videau, "Symmetric boolean functions," *IEEE Transactions on Information Theory*, vol. 51, no. 8, pp. 2791–2811, 2005.

[9] I. Wegener, *The Complexity of Boolean Functions*. USA: John Wiley Sons, Inc., 1987.

[10] H. Rahaman, D. K. Das, and B. B. Bhattacharya, "A new synthesis of symmetric functions," in *Proceedings of ASP-DAC/VLSI Design 2002. 7th Asia and South Pacific Design Automation Conference and 15h International Conference on VLSI Design*, 2002, pp. 160–165.

[11] H. Rahaman and D. K. Das, "A simple delay testable synthesis of symmetric functions," in *Applied Computing*, S. Manandhar, J. Austin, U. Desai, Y. Oyanagi, and A. K. Talukder, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 263–270.

[12] A. Deb, D. K. Das, and B. B. Bhattacharya, "Synthesis of symmetric boolean functions using a three-stage network," in *2014 Fifth International Symposium on Electronic System Design*, 2014, pp. 182–186.

[13] I. Gunawan, "Reliability analysis of shuffle-exchange network systems," *Reliability Engineering System Safety*, vol. 93, no. 2, pp. 271–276, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832007000452

[14] C. Wolf, "Yosys - yosys open synthesis suite." [Online]. Available: http://www.clifford.at/yosys/about.html