electronics

Article

Area, Power and Speed Optimized Early Output Majority Voter for Asynchronous TMR Implementation

Padmanabhan Balasubramanian [1,*] and Nikos E. Mastorakis [2]

[1] School of Computer Science and Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798, Singapore
[2] Department of Industrial Engineering, Technical University of Sofia, 1000 Sofia, Bulgaria; mastor@tu-sofia.bg
[*] Correspondence: balasubramanian@ntu.edu.sg; Tel.: +65-679-047-45

Abstract: This paper presents a new, efficient asynchronous early output majority voter that can be used to effectively realize an asynchronous triple modular redundancy (TMR) implementation. For the input-output mode asynchronous realization, the dual-rail code was used for data encoding and four phase return-to-zero and return-to-one handshake schemes were separately used for data communication. The proposed majority voter requires 62.8% less area and dissipates 37% less power on average compared to the best of the existing asynchronous majority voters while considering both handshake schemes. Importantly, the reductions in area and power are achieved without sacrificing the speed. Example TMR implementations show that the proposed majority voter leads to simultaneous reductions in cycle time, silicon area, and power dissipation. As a result, the proposed majority voter enables improved optimization in figure-of-merits such as area–cycle time product, power–cycle time product, and area–cycle time–power product for TMR implementations utilizing it compared to TMR implementations incorporating other majority voters. The circuits were implemented using a 32/28-nm CMOS technology.

Keywords: digital circuits; logic design; fault tolerance; low power; high speed; TMR; CMOS

Citation: Balasubramanian, P.; Mastorakis, N.E. Area, Power and Speed Optimized Early Output Majority Voter for Asynchronous TMR Implementation. *Electronics* **2021**, *10*, 1425. https://doi.org/10.3390/electronics10121425

Academic Editor: Antonio G. M. Strollo

Received: 15 May 2021
Accepted: 11 June 2021
Published: 14 June 2021

1. Introduction

In many mission- and safety-critical applications such as space, aerospace, nuclear power plants, electric power transmission and distribution, banking and stock exchanges, and industrial control and automation, etc. N-modular redundancy is widely used, with triple modular redundancy (TMR) being common [1,2]. To realize a TMR implementation, three identical function blocks are used, where a function block may be a circuit or a sub-system or a system. The corresponding output(s) of the function blocks are majority voted using majority voter(s). Suppose a function block produces K outputs, K 3-input majority voters are required to realize a TMR implementation [3].

Depending upon the design paradigm adopted, the majority voter used in a TMR implementation may be realized in a synchronous or asynchronous design style. Different designs of majority voters for a synchronous TMR implementation are available in the literature [4–6]. Different designs of majority voters for an asynchronous TMR implementation are also available in the literature [7–10]. Reference [7] corresponds to a bundled data asynchronous design while the studies in [8–10] correspond to an input-output mode asynchronous design. Generally, an input-output mode asynchronous design is more robust than a bundled data asynchronous design since the former is not delay-sensitive while the latter is sensitive to delays [11]. Note that henceforth, by 'asynchronous realization' we mean an 'input-output mode asynchronous realization' in this paper unless stated otherwise.

In [8,9], strongly indicating majority voters for an asynchronous TMR implementation were presented while in [10] an early output majority voter was presented. This paper presents a new, improved early output majority voter for an asynchronous TMR implementation. The dual-rail code is used for data encoding and four-phase return-to-zero handshaking (RTZH) and return-to-one handshaking (RTOH) schemes are separately used for data communication.

The rest of this paper is organized as follows. Section 2 briefly discusses fundamentals of input-output mode asynchronous circuits. Section 3 presents the proposed early output majority voter and describes how it is used to realize an efficient asynchronous TMR implementation. Section 4 gives the design metrics of example TMR implementations which utilize different majority voters including the proposed majority voter. Section 5 gives the conclusions.

2. Input-Output Mode Asynchronous Circuit

A typical block diagram of an input-output mode asynchronous pipeline stage is shown in Figure 1a. It consists of a pair of input and output register banks, control signals (AIN and AOUT) which are exchanged between the register banks to control the data flow, an asynchronous circuit that is sandwiched between the register banks, and completion detectors.

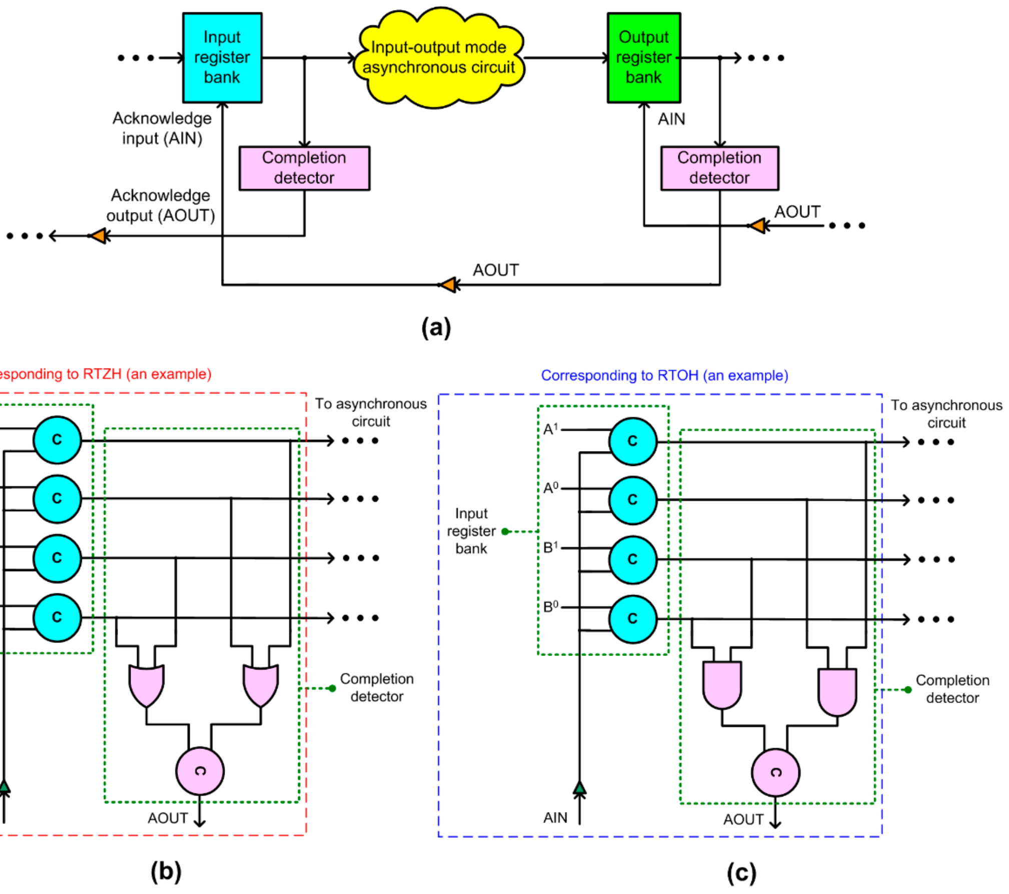


Figure 1. (**a**) Block diagram of an input-output mode asynchronous circuit stage. Example illustration of input register bank and completion detector corresponding to: (**b**) RTZH and (**c**) RTOH.

In Figure 1a, an input register bank and an output register bank are shown. The input register bank could serve as an output register bank for a preceding stage in the pipeline, and the output register bank could serve as an input register bank for a successive stage in the pipeline. Supposing that dual-rail encoded signals viz. (A^1,A^0) and (B^1,B^0) are supplied through the input register bank, a two-input C-element is used for each rail of every dual-rail encoded input which is synchronized with the acknowledgment input signal (AIN), and forwarded to the asynchronous circuit for processing, as depicted in Figure 1b,c. This description is applicable for the output register bank as well, shown in Figure 1a. To avoid the loading on AIN signal, a non-inverting buffer or buffer tree may be used.

The C-element is a rendezvous element that is often used to construct asynchronous circuits. If all the inputs to a C-element are 1 or 0, it would output 1 or 0, respectively. On the other hand, if any of the inputs to a C-element is different, the C-element would retain its existing steady state. A static CMOS realization of a 2-input C-element [12] is shown in Figure 2, where P and Q are the inputs and R is the output. In Figure 1b,c, the circles with the marking 'C' represent the C-element.

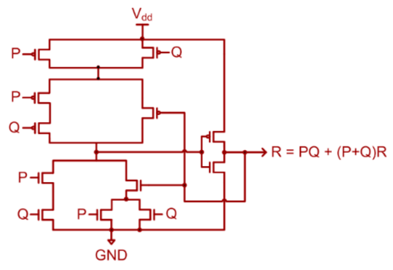


Figure 2. Static CMOS realization of a two-input Muller C-element.

AIN and AOUT represent acknowledgment input and output control signals in Figure 1, which are Boolean complements of each other. AIN and AOUT are used for data communication, i.e., for handshaking between the input and output register banks, and they control the data flow in an input-output mode asynchronous pipeline stage. Dual-rail encoding of inputs and outputs is performed according to the handshake scheme used. For example, a signal A is encoded into dual-rail signals (A^1,A^0) based on RTZH and RTOH, as shown in Table 1.

Table 1. Dual-rail data encoding based on four phase RTZH and RTOH.

Handshake Scheme	**Single Rail Input (A)**	**Dual-Rail Equivalent**		**State Definition**
		$\mathbf{A^1}$	$\mathbf{A^0}$	
RTZH	0	0	1	Binary 0
	1	1	0	Binary 1
	–	0	0	Spacer
	–	1	1	Indeterminate
RTOH	0	1	0	Binary 0
	1	0	1	Binary 1
	–	1	1	Spacer
	–	0	0	Indeterminate

In Table 1, with respect to RTZH [11], $A^1 = 1$ and $A^0 = 0$, and $A^0 = 1$ and $A^1 = 0$ are referred to as *data*. Only one of the rails of each dual-rail encoded input is asserted high to represent data, and binary 1 is used to represent data in the case of RTZH. $A^1 = A^0 = 0$ is referred to as the *spacer* that is supplied between two data inputs, and $A^1 = A^0 = 1$ is designated as indeterminate or illegal.

With respect to RTOH [13], $A^1 = 0$ and $A^0 = 1$, and $A^0 = 0$ and $A^1 = 1$ are referred to as *data*. Only one of the rails of each dual-rail encoded input is asserted low to represent data, and binary 0 is used to represent data in the case of RTOH. $A^1 = A^0 = 1$ is referred to as the *spacer* that is supplied between two data inputs, and $A^1 = A^0 = 0$ is deemed indeterminate.

Four phases or steps are involved in RTZH and RTOH. In the case of RTZH, with AIN being 1 initially, a fresh data can be input to the asynchronous circuit for processing, and after this, AOUT would assume 1 through the completion detector. Subsequently, AIN would assume 0. With AIN = 0, the spacer is supplied to the asynchronous circuit for processing, and eventually AOUT will assume 0 through the completion detector and AIN would assume 1 again. Another data sequence can then be input to the asynchronous circuit for processing. These imply that the application of inputs according to RTZH follows the sequence *data-spacer-data-spacer* and so on, i.e., the data bus would entirely assume zero after the application of an input data.

In the case of RTOH, with AIN being 1 initially, the spacer is supplied to the asynchronous circuit for processing, and AOUT would assume 1 through the completion detector. Following this, AIN would assume 0, and a fresh data can be input to the asynchronous circuit for processing. Subsequently, AOUT would assume 0, implying that AIN will assume 1 again and the spacer can be supplied. These imply that, according to RTOH, the application of inputs follows the sequence *spacer-data-spacer-data* and so on, i.e., the data bus entirely assumes one after the application of an input data.

Example completion detectors corresponding to RTZH and RTOH are shown in Figure 1b,c, respectively, assuming that single rail primary inputs A and B are dual-rail encoded as (A^1,A^0) and (B^1,B^0). The dual rails of each encoded primary input is combined using a two-input OR gate in the case of RTZH and a two-input AND gate in the case of RTOH. The outputs of two-input OR (AND) gates are combined using a C-element in the case of RTZH (RTOH) to produce the completion detector output. When many two-input OR gates or two-input AND gates are present, their outputs can be combined using a tree of C-elements to produce the completion detector output. The output of the completion detector acknowledges the receipt of all the primary inputs given to the asynchronous circuit for processing.

3. Proposed Early Output Majority Voter and Asynchronous TMR Implementation

The block diagram of an asynchronous TMR implementation is shown in Figure 3, with asynchronous function blocks 1, 2 and 3 all being identical. Dual-rail encoding is used and four-phase RTZH or RTOH may be used. At least two out of the three function blocks should always maintain the correct operation. In Figure 3, (A^1,A^0), (B^1,B^0) and (C^1,C^0) represent the dual-rail primary inputs given to asynchronous function blocks 1, 2 and 3, and (X^1,X^0), (Y^1,Y^0) and (Z^1,Z^0) are their corresponding primary outputs which are majority voted to generate the voter's primary output (M^1,M^0).

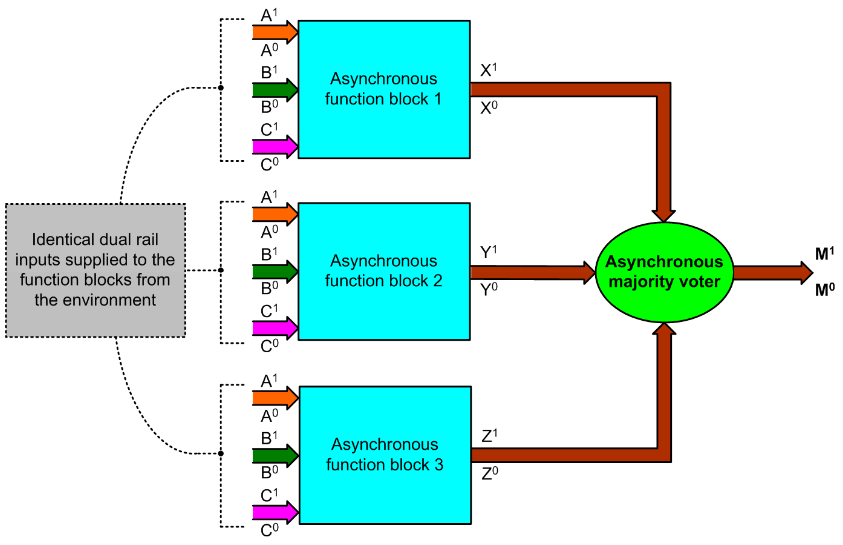

Identical dual rail inputs supplied to the function blocks from the environment
A¹
A⁰
B¹
B⁰
C¹
C⁰
Asynchronous function block 1
Asynchronous function block 2
Asynchronous function block 3
X¹
X⁰
Y¹
Y⁰
Z¹
Z⁰
Asynchronous majority voter
M¹
M⁰

Figure 3. Asynchronous TMR implementation utilizing dual-rail encoding.

The logic expressions of the majority voter outputs corresponding to RTZH are given by Equations (1) and (2), where the first three terms on the right side of the equations signify the correct operation of at least two function blocks and the last term specifies the correct operation of all the function blocks.

$$M^1 = X^1Y^1 + Y^1Z^1 + X^1Z^1 + X^1Y^1Z^1 \tag{1}$$

$$M^0 = X^0Y^0 + Y^0Z^0 + X^0Z^0 + X^0Y^0Z^0 \tag{2}$$

The logic expressions of the majority voter outputs corresponding to RTOH are given by Equations (3) and (4). In general, the dual of the logic expressions corresponding to RTZH would give rise to logic expressions that adhere to RTOH, and vice-versa. It may be recalled that binary 1 is used to represent data in the case of RTZH with zeroes used as the spacer and binary 0 is used to represent data in the case of RTOH with ones used as the spacer.

$$M^1 = (X^1 + Y^1)\,(Y^1 + Z^1)\,(X^1 + Z^1)\,(X^1 + Y^1 + Z^1) \tag{3}$$

$$M^0 = (X^0 + Y^0)\,(Y^0 + Z^0)\,(X^0 + Z^0)\,(X^0 + Y^0 + Z^0) \tag{4}$$

The truth table of the dual-rail encoded majority voter corresponding to RTZH and RTOH is given in Table 2.

Table 2. Truth table of the dual-rail encoded majority voter (showing only data and no spacer).

Dual-Rail Encoded Inputs						**Dual-Rail Encoded Outputs**	
X^1	X^0	Y^1	Y^0	Z^1	Z^0	M^1	M^0
Based on RTZH							
0	1	0	1	0	1	0	1
0	1	0	1	1	0	0	1
0	1	1	0	0	1	0	1
0	1	1	0	1	0	1	0
1	0	0	1	0	1	0	1
1	0	0	1	1	0	1	0
1	0	1	0	0	1	1	0
1	0	1	0	1	0	1	0
Based on RTOH							
1	0	1	0	1	0	1	0
1	0	1	0	0	1	1	0
1	0	0	1	1	0	1	0
1	0	0	1	0	1	0	1
0	1	1	0	1	0	1	0
0	1	1	0	0	1	0	1
0	1	0	1	1	0	0	1
0	1	0	1	0	1	0	1

The proposed early output majority voter is shown in Figure 4, where Figure 4a corresponds to RTZH and Figure 4b corresponds to RTOH. To transform a circuit that corresponds to RTZH into one that corresponds to RTOH and vice versa, excepting the C-elements, the rest of the gates should be replaced by their respective duals. This property was proven through induction in [14].

Figure 4a comprises two complex gates and Figure 4b also comprises two complex gates. An AO222 or an OA222 complex gate is realized in static CMOS style using 12 transistors. Hence, Figure 4a,b would require just 24 transistors each for a static CMOS implementation. Figure 4a,b perhaps represent the ultimate gate-level design of an early output asynchronous majority voter based on dual-rail encoding which correspond to RTZH and RTOH respectively. This is because Figure 4a,b feature only one gate for each rail of the encoded voter output.

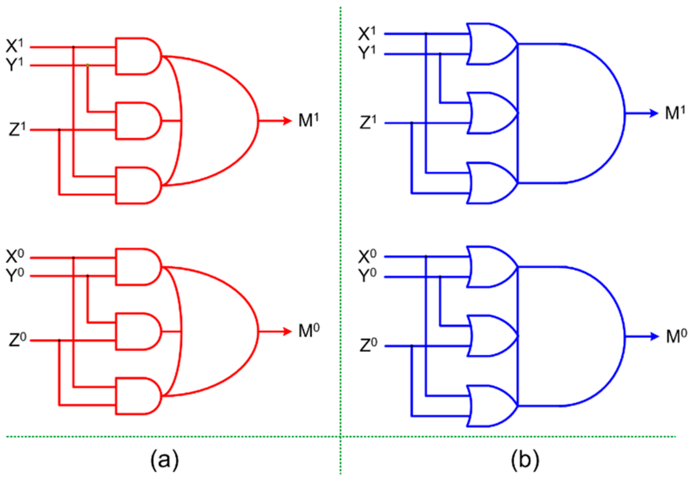


Figure 4. Proposed early output majority voter corresponding to: (**a**) RTZH; (**b**) RTOH.

The proposed early output majority voter is a monotonic Boolean network (MBN) [15]. Referring to Figure 3, the asynchronous TMR implementation consisting of identical function blocks and the proposed majority voter(s) is also an MBN. In an MBN, the signal transitions monotonically increase or decrease from the primary inputs up to the primary outputs subsequent to the application of inputs [16]. In an MBN corresponding to RTZH, the signal transitions monotonically increase from binary 0 to 1 for the application of data, and monotonically decrease from binary 1 to 0 for the application of spacer. For example, after the application of spacer in Figure 4a, if data are applied whereby X^0, Y^0 and Z^0 assume 1, then M^0 would assume 1. Subsequently, in the following return-to-zero phase, if X^0, Y^0 and Z^0 assume 0, M^0 would also assume 0.

On the other hand, in an MBN corresponding to RTOH, the signal transitions monotonically decrease from binary 1 to 0 for the application of data, and monotonically increase from binary 0 to 1 for the application of spacer. For example, after the application of spacer in Figure 4b, if data are applied whereby X^0, Y^0 and Z^0 assume 0, then M^0 would assume 0. Subsequently, in the following return-to-one phase, if X^0, Y^0 and Z^0 assume 1, M^0 would also assume 1.

An early output asynchronous circuit [17] is able to commence the processing after receiving just a subset of the primary inputs and can produce all the primary output(s). The early output nature of the proposed majority voter shown in Figure 4a is described by an example scenario as follows. After the application of data, $X^1 = Y^1 = 1$ could result in $M^1 = 1$ early in accordance with the Boolean majority, regardless of Z^1 having assumed 1. Subsequently, in the following return-to-zero phase after the application of spacer, if X^1 and Y^1 assume 0, M^1 could assume 0 early regardless of Z^1 or Z^0 having assumed 0. Thus, the early output nature may be manifested for the application of data and/or spacer in Figure 4a.

The early output nature of the proposed majority voter shown in Figure 4b is described by an example scenario as follows. With M^1 and M^0 being 1 initially (due to the application of spacer), if X^1 and Y^1 assume 0 after the application of data, M^1 could assume 0 early regardless of Z^1 having assumed 0. Subsequently, in the following return-to-one phase, if X^1 and Y^1 assume 1, M^1 could assume 1 early, regardless of Z^1 having assumed 1. Thus, the early output nature may be manifested for the application of data and/or spacer in Figure 4b.

The proposed early output majority voter is different from the majority voters presented in [8,9], which are strongly indicating. A strongly indicating asynchronous circuit [18] requires all the primary inputs to process and produce all the primary output(s). This implies that a strong indication majority voter would require all the primary inputs to process them and produce data as well as spacer. Due to this strict timing constraint, strong indication asynchronous circuits generally have inferior design metrics compared to other categories of asynchronous circuits [14]. Although the majority voter presented in [10] is of early output type, similar to our proposed majority voter, however, it contains relatively more logic and hence our proposed majority voter facilitates better optimization of the design metrics, which is discussed in the next section.

4. Results

TMR circuits incorporating existing majority voters [8–10] and the proposed majority voter called "NEO_MV" were implemented adhering to RTZH and RTOH, separately. A full adder [19] was used to represent the function blocks. Excepting the 2-input C-element that was custom realized, as shown in Figure 2, the rest of the gates were directly utilized from the 32/28 nm CMOS standard digital cell library [20]. A typical case process, voltage and temperature specification corresponding to a low leakage (high V_t) process with a supply voltage of 1.05V and an operating junction temperature of 25 °C was considered for the simulations. Default wire loads were included, and a fan-out of 4 drive strength was assigned to all the output ports, i.e., majority voter outputs.

Different TMR implementations were simulated, and their functionality was verified by considering all the distinct inputs for the function blocks, which were supplied through a test bench. The test bench indirectly modelled the presence and absence of fault(s) in the function block(s). In other words, all the distinct combinations of the majority voter's primary inputs shown in Table 2 were generated through the outputs of function blocks to perform functional simulations of the majority voters. For example, referring to Table 2, with respect to RTZH, $X^1 = Y^1 = Z^1 = 1$ and $X^0 = Y^0 = Z^0 = 1$ denote those instances when all the function blocks operate correctly and produce the same outputs. The remaining instances represent the scenarios when one of the function blocks is faulty, but a majority of the function blocks still operate correctly. Likewise, with respect to RTOH, referring to Table 2, $X^1 = Y^1 = Z^1 = 0$ and $X^0 = Y^0 = Z^0 = 0$ refer to those instances when all the function blocks operate correctly and produce the same outputs. The remaining instances represent the scenarios when one of the function blocks is faulty. However, a majority of the function blocks still operate correctly. The test bench was supplied at a latency of 2 ns, i.e., at a cycle time of 4 ns. The switching activity data obtained from the simulations were used to estimate the total power dissipation.

Screenshots of portions of simulation waveforms of TMR implementations utilizing the proposed majority voter (NEO_MV) are shown in Figures 5 and 6, corresponding to RTZH and RTOH, respectively. In Figures 5 and 6, (SUM21,SUM20), (SUM11,SUM10) and (SUM01,SUM00) represent the sum outputs of triplicated full adders while (CARRY21,CARRY20), (CARRY11,CARRY10) and (CARRY01,CARRY00) represent the carry outputs of the full adders. The majority voted sum and carry outputs are denoted by (SUM1,SUM0) and (CARRY1,CARRY0) in Figures 5 and 6, whose waveforms are highlighted in blue. The waveforms in Figures 5 and 6 show that correct majority voted outputs are produced in accordance with the Boolean majority despite the faulty behavior of a function block.

In an input-output mode asynchronous circuit, the cycle time is an important metric that governs the operating speed. The cycle time is the time duration to complete one transaction that involves the processing of a data and the spacer. The time taken to process the data and the spacer are called forward latency and reverse latency respectively, and the cycle time is the sum of these latencies. For the TMR implementations considered, their forward and reverse latencies are equal, and so the cycle time is equal to a doubling of the forward latency/reverse latency.

The standard design metrics such as area, cycle time, and power dissipation were estimated for different TMR implementations corresponding to RTZH and RTOH, and they are given in Table 3. The differences in the design metrics of TMR implementations are entirely attributable to the differences in the logic of the majority voters, since the function blocks, registers and completion detector used are the same.

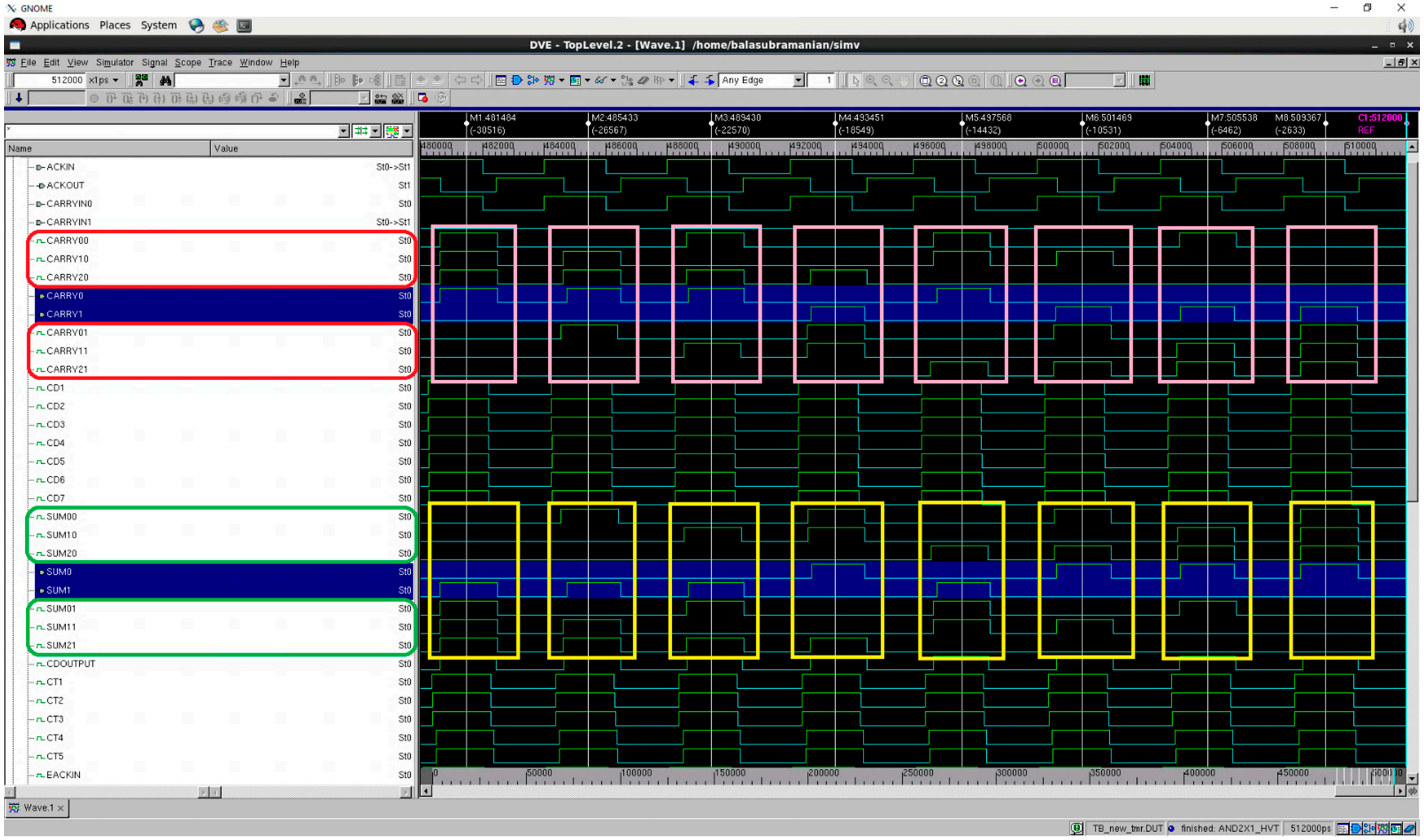
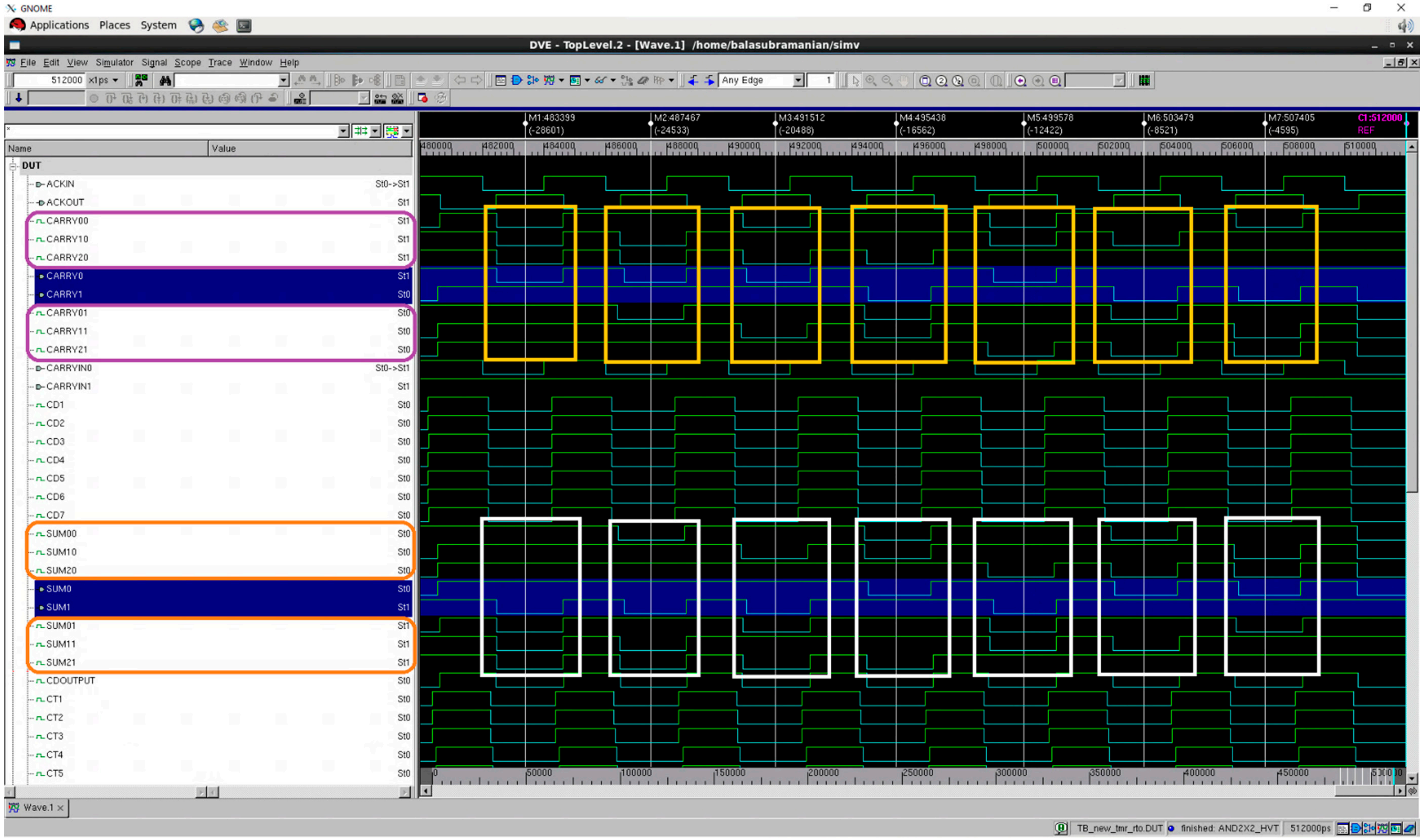


Figure 5. Screenshot of a portion of simulation waveforms of the TMR implementation incorporating the proposed majority voter (NEO_MV) corresponding to RTZH. Binary 1 represents data in the case of RTZH. Majority voted sum output (SUM1,SUM0) is captured within the yellow rectangles and majority voted carry output (CARRY1,CARRY0) is captured within the rose rectangles.


GNOME
Applications Places System
DVE - TopLevel.2 - [Wave.1] /home/balasubramanian/simv
File Edit View Simulator Signal Scope Trace Window Help
512000 x1ps
Any Edge
M1:483399 (-28601)
M2:487467 (-24533)
M3:491512 (-20488)
M4:495438 (-16562)
M5:499578 (-12422)
M6:503479 (-8521)
M7:507405 (-4595)
C1:512000 REF
Wave.1
TB_new_tmr_rto.DUT finished: AND2X2_HVT 512000ps

Name	Value
DUT	
ACKIN	St0->St1
ACKOUT	St1
CARRY00	St1
CARRY10	St1
CARRY20	St1
CARRY0	St1
CARRY1	St0
CARRY01	St0
CARRY11	St0
CARRY21	St0
CARRYIN0	St0->St1
CARRYIN1	St1
CD1	St0
CD2	St0
CD3	St0
CD4	St0
CD5	St0
CD6	St0
CD7	St0
SUM00	St0
SUM10	St0
SUM20	St0
SUM0	St0
SUM1	St1
SUM01	St1
SUM11	St1
SUM21	St1
CDOUTPUT	St0
CT1	St0
CT2	St0
CT3	St0
CT4	St0
CT5	St0

Figure 6. Screenshot of a portion of simulation waveforms of the TMR implementation incorporating the proposed majority voter (NEO_MV) corresponding to RTOH. Binary 0 represents data in the case of RTZH. Majority voted sum output (SUM1,SUM0) is captured within the white rectangles and majority voted carry output (CARRY1,CARRY0) is captured within the yellow rectangles.

Table 3. Design metrics of asynchronous TMR implementations with respect to RTZH and RTOH.

Majority Voter Used	Area (μm^2)	Cycle Time (ns)	Power Dissipation (μW)		
			Majority Voters	Others *	Total
		Based on RTZH			
Singh_MV [8]	327.08	2.84	25.29	128.91	154.2
DIMS_MV [8]	319.46	2.30	15.94	130.06	146.0
Toms_MV [8]	312.34	2.40	16.37	129.43	145.8
SI_MV [8]	300.14	2.50	27.26	128.74	156.0
CG_MV [9]	277.78	2.46	24.94	128.26	153.2
EO_MV [10]	261.51	1.74	6.50	127.70	134.2
NEO_MV (proposed)	**239.15**	**1.70**	**4.11**	127.29	**131.4**
		Based on RTOH			
Singh_MV [8]	327.08	2.78	25.43	129.77	155.2
DIMS_MV [8]	315.39	2.24	14.74	130.96	145.7
Toms_MV [8]	312.34	2.38	16.71	130.39	147.1
SI_MV [8]	300.14	2.48	27.17	129.93	157.1
CG_MV [9]	277.78	2.46	24.76	128.94	153.7
EO_MV [10]	261.51	1.74	6.39	129.01	135.4
NEO_MV (proposed)	**239.15**	**1.66**	**4.01**	128.09	**132.1**

* Refers to function blocks, registers and completion detector.

Overall, from Table 3, we see that the proposed NEO_MV enables better reductions in area, cycle time, and power dissipation compared to its counterparts. This is due to two main reasons. Considering both RTZH and RTOH, on average, the areas of the majority voters are found as: Singh_MV–50.57μm^2; DIMS_MV–45.75 μm^2; Toms_MV–43.20 μm^2; SI_MV–37.11 μm^2; CG_MV–25.92 μm^2; EO_MV–17.79 μm^2; and the proposed NEO_MV–6.61 μm^2. Therefore, NEO_MV requires 62.8% less area compared to the best of the existing majority voters viz. EO_MV. The lesser area occupancy of NEO_MV leads to lesser power dissipation, and its latency is less because it has a single logic level. The split-up of power dissipated by majority voters and the power dissipated by function blocks, registers and completion detector is also given in Table 3. The sum of these power components gives the total power dissipation. From Table 3, we note that compared to EO_MV, NEO_MV dissipates 36.8% less power for RTZH and 37.2% less power for RTOH, i.e., 37% less power on average. Overall, NEO_MV enables reduced total power dissipation for the TMR implementations comprising it compared to TMR implementations incorporating the other majority voters.

Besides estimating the standard design metrics such as cycle time, area and average power dissipation, some figure-of-merits (FOMs) are also calculated for VLSI designs. It is desirable to have reduced design metrics which is characteristic of an effective circuit design. Given this, the FOMs are a combined interpretation of the design metrics. Some popular FOMs include area–delay product (here, area–cycle time product (ACTP)), power-delay product (here, power–cycle time product (PCTP)), and area–delay–power product (here, area–cycle time–power product (ACTPP)). Among these, PCTP is representative of energy.

We calculated the actual values of the above mentioned FOMs for all the TMR implementations and then normalized them. To normalize, the highest value of actual FOM of a TMR implementation with respect to a specific handshake scheme was considered as the baseline and this was used to divide the actual FOM of all the TMR implementations.

This procedure was followed to normalize the FOMs of various TMR implementations corresponding to RTZH and RTOH separately, and the normalized FOMs are portrayed by Figure 7a,b. A smaller value is preferable for a normalized FOM, which is indicative of an effective circuit design.

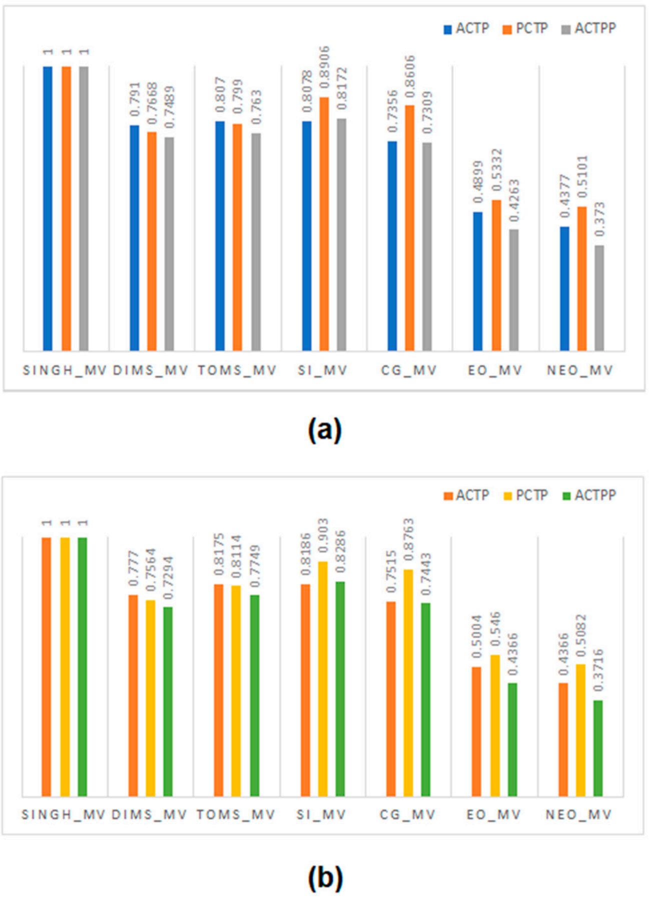


Figure 7. Normalized FOMs of TMR implementations utilizing different majority voters corresponding to: (**a**) RTZH; and (**b**) RTOH.

In Figure 7a, the blue, orange and grey bars denote normalized ACTP, PCTP and ACTPP, respectively, corresponding to RTZH, and in Figure 7b, the orange, yellow and green bars denote the same order of normalized FOMs with respect to RTOH. It is clear from Figure 7a,b that TMR implementations utilizing the proposed majority voter (NEO_MV) achieve better optimized FOMs compared to TMR implementations incorporating the other majority voters. This is due to the reduced design metrics reported for NEO_MV compared to its counterparts, as noted from Table 3.

From Figure 7a,b, and Table 3, it may be inferred that among the existing majority voters, EO_MV is optimized. However, the proposed NEO_MV is better optimized than EO_MV. TMR implementations incorporating NEO_MV achieve the following savings in FOMs compared to TMR implementations incorporating EO_MV: (i) 10.7% reduction in ACTP, 4.1% reduction in PCTP (energy) and 12.5% reduction in ACTPP with respect to RTZH; and (ii) 12.7% reduction in ACTP, 6.9% reduction in PCTP (energy) and 14.9% reduction in ACTPP with respect to RTOH.

5. Conclusions

This paper presented a new compact early output majority voter that is useful for realizing an effective asynchronous TMR implementation. The dual-rail code was used for data encoding and four-phase handshake schemes were used for data communication.

Different TMR circuits were implemented incorporating the proposed majority voter and the other existing majority voters. The design metrics report enhanced reductions in area, cycle time, power dissipation, and thus reduced FOMs for TMR implementations incorporating the proposed majority voter compared to TMR implementations incorporating other existing majority voters.

Author Contributions: Conceptualization, P.B. and N.E.M.; methodology, P.B.; software, P.B.; validation, P.B.; formal analysis, P.B. and N.E.M.; investigation, P.B.; data curation, P.B.; visualization, P.B.; writing—original draft preparation, P.B.; writing—review and editing, P.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

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

References

1. Johnson, B.W. *Design and Analysis of Fault-Tolerant Digital Systems*; Addison-Wesley: Boston, MA, USA, 1989; ISBN 978-0201075700.
2. Koren, I.; Krishna, C.M. *Fault-Tolerant Systems*; Morgan Kaufmann Publishers: Burlington, MA, USA, 2007; ISBN 978-0120885251.
3. Choudhary, J.; Balasubramanian, P.; Varghese, D.M.; Singh, D.P.; Maskell, D. Generalized majority voter design method for N-modular redundant systems used in mission- and safety-critical applications. *Computers* **2019**, *8*, 10. [CrossRef]
4. Kshirsagar, R.V.; Patrikar, R.M. Design of a novel fault-tolerant voter circuit for TMR implementation to improve reliability in digital circuits. *Microelectron. Reliab.* **2009**, *49*, 1573–1577. [CrossRef]
5. Ban, T.; Naviner, L.A.B. A simple fault-tolerant digital voter circuit in TMR nanoarchitectures. In Proceedings of the 8th IEEE International NEWCAS Conference, Montreal, QC, Canada, 20–23 June 2010.
6. Balasubramanian, P.; Prasad, K. A fault tolerance improved majority voter for TMR system architectures. *WSEAS Trans. Circuits Syst.* **2016**, *15*, 108–122.
7. Almukhaizim, S.; Sinanoglu, O. Novel hazard-free majority voter for N-modular redundancy-based fault tolerance in asynchronous circuits. *IET Comput. Digit. Tech.* **2011**, *5*, 306–315. [CrossRef]
8. Balasubramanian, P.; Maskell, D.L.; Mastorakis, N.E. Quasi delay insensitive majority voters for triple modular redundancy applications. *Appl. Sci.* **2019**, *9*, 5400. [CrossRef]
9. Balasubramanian, P.; Maskell, D.L.; Mastorakis, N.E. Area optimized quasi delay insensitive majority voter for TMR applications. In Proceedings of the IEEE 3rd European Conference on Electrical Engineering and Computer Science, Athens, Greece, 28–30 December 2019.
10. Balasubramanian, P.; Maskell, D.L.; Mastorakis, N.E. Asynchronous early output majority voter and a relative-timed asynchronous TMR implementation. *Microelectron. Reliab.* **2020**, *114*, 1–5. [CrossRef]
11. Sparsø, J.; Furber, S.B. *Principles of Asynchronous Circuit Design: A Systems Perspective*; Kluwer Academic Publishers: Dordrecht, The Netherlands, 2001; ISBN 978-0792376132.
12. Beerel, P.A.; Ozdag, R.O.; Ferretti, M. *A Designer's Guide to Asynchronous VLSI*; Cambridge University Press: Cambridge, UK, 2010; ISBN 978-0521872447.
13. Moreira, M.T.; Guazzelli, R.A.; Calazans, N.L.V. Return-to-one protocol for reducing static power in C-elements of QDI circuits employing m-of-n codes. In Proceedings of the 25th Symposium on Integrated Circuits and Systems Design, Brasilia, Brazil, 30 August–2 September 2012.
14. Balasubramanian, P. Comparative evaluation of quasi-delay-insensitive asynchronous adders corresponding to return-to-zero and return-to-one handshaking. *Facta Univ. Ser. Electron. Energetics* **2018**, *31*, 25–39. [CrossRef]
15. Cortadella, J.; Kondratyev, A.; Lavagno, L.; Sotiriou, C. Coping with the variability of combinational logic delays. In Proceedings of the IEEE International Conference on Computer Design, San Jose, CA, USA, 11–13 October 2004.
16. Varshavsky, V.I. Aperiodic circuits. In *Self-Timed Control of Concurrent Processes: The Design of Aperiodic Logical Circuits in Computers and Discrete Systems*; Varshavsky, V.I., Ed.; (Translated from Russian by Yakovlev, A.V.); Kluwer Academic Publishers: New York, NY, USA, 1990; pp. 77–85.
17. Brej, C. Early Output Logic and Anti-Tokens. Ph.D. Thesis, The University of Manchester, Manchester, UK, September 2005.
18. Seitz, C.L. System timing. In *Introduction to VLSI Systems*; Mead, C., Conway, L., Eds.; Addison-Wesley: Reading, MA, USA, 1980; pp. 218–262. ISBN 978-0201043587.
19. Toms, W.B.; Edwards, D.A. Efficient synthesis of speed-independent combinational logic circuits. In Proceedings of the Asia and South Pacific Design Automation Conference, Shanghai, China, 18–21 January 2005.
20. Synopsys SAED_EDK32/28_CORE Databook. Revision 1.0.0. January 2012. Available online: https://www.synopsys.com/community/university-program/teaching-resources.html (accessed on 12 March 2021).