High-level Modeling and Verification Platform for Elastic Circuits with Process Variation Considerations

MEYSAM ZAEEMI[*]

School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran,
zaeemi@ut.ac.ir

SIAMAK MOHAMMADI[*,a,b]

[a]School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran,

[b]School of Computer Science, Institute of Fundamental Sciences (IPM), Tehran, Iran

smohamadi@ut.ac.ir

In addition to the advantages of asynchronous circuits, compatibility with synchronous EDA tools is another strength point of synchronous elastic circuits. Synchronous elastic circuits face some challenges, such as process variations that can compromise its performance and functionality, and the multitude of available implementations based on elastic elements' combinations, meaning that choosing the best combination could not be simple. In this paper, a novel method is introduced to model and verify synchronous elastic circuits in the presence of variations. The model is based on xMAS, which is a new formal modeling paradigm to synthesize, test, and verify circuits and networks. In this method, various elastic elements are modeled and available in the form of a library in xMAS, so the designer can build complicated elastic circuits by combining different elastic elements. Additionally, by translating high-level xMAS model into SAN statistical model and using its capabilities, elements' internal delays will be embedded, which makes the high-level modeling and elastic circuits' high-resolution time analysis available. Based on the obtained results, elastic circuits are highly capable of tolerating variations. However, this phenomenon could lead to a maximum of 2.35% error in synchronization control units and data in these circuits.

CCS CONCEPTS •Hardware~Hardware validation~Functional verification•Mathematics of computing~Probability and statistics~Distribution functions•Computing methodologies~Modeling and simulation~Model development and analysis~Modeling methodologies

Additional Keywords and Phrases: High-level modeling, Synchronous elastic circuit, Timing verification, Process variation

1 INTRODUCTION

The synchronous design approach, which has achieved complete maturity in Electronic Design Automation (EDA) tools, has been known for many years as the first and best option to design circuits. With technology scaling, on-chip clock distribution in a synchronous approach has encountered difficulties that have led some researchers to turn their focus on asynchronous design in digital circuits. Despite all the improvements, and significant advantages in terms of low energy consumption, high operating speed and variation tolerance, due to the lack of commercial EDA tools, asynchronous design has not been widely used by designers. The synchronous elastic design has emerged to fill the gap between synchronous and asynchronous designs. This paradigm can be designed based on standard synchronous EDA, and at the same time, benefit from some of the asynchronous design advantages such as latency variations tolerance in the computations and communications. Elastic designs have been studied for various applications. They have shown excellent performance characteristics, such as elastic structure in

[*] University of Tehran, School of Electrical and Computer Engineering, North Kargar Ave, Tehran, Iran. PO Box: 14395/515

1550-4832/2022/1-ART1 $15.00
https://doi.org/10.1145/3534971

flow control of network on chip (NOC) [1, 2], dataflow networks [3], globally asynchronous locally synchronous (GALS) structure [4, 5], dynamic scheduling in High-level Synthesis [6] and elastic silicon interconnect [7]. Although many studies have focused on designing elastic circuits, results on elastic structures verification are limited. Designing elastic systems is error-prone, and one must verify that the designed structure complies with the specified properties. The primary purpose of many studies in elastic system verification research is to validate elastic protocols [8-11]. The authors in [12] provides correctness proofs for methods that are used to synthesize elastic designs from synchronous designs, although these correctness proofs do not replace verification. In [13], the authors proposed a refinement technique to verify that an elastic circuit is equivalent to its synchronous design. Since in the data flow of elastic architectures, buffers could be used in any place and any number to optimize the performance, using refinement methods leads to complications in generating refinement maps.

With technology scaling, process variation has become a challenging factor in circuit design. Variability affects digital circuits' functionality and performance[14, 15] which means that the characteristics of the manufactured circuit could be different from designed characteristics. Although elastic circuits show more tolerance against variations compared to synchronous designs, the variability can still compromise functionality, and performance in this type of circuit. Few research works have examined its effects on elastic circuits. In [16], elastic link pipelining and buffers under process variation have been analyzed. However, other elements such as different types of fork and join have never been studied before.

Raising the abstraction level means that the designers will have a higher chance of choosing the best design by architectural exploration. xMAS is one of the most recent high-level modeling platforms introduced by Intel researchers [17]. Using xMAS could eliminate a large number of common unintentional errors. Additionally, xMAS exceeds other tools used for verification in terms of performance and scalability. xMAS has recently been used for modeling and verification in different studies [18-21]; however, statistical information has never been used with the goal of increasing the precision in the high level modeling.

In this paper, all elastic elements are evaluated and compared, by taking into account variability. Internal delays are calculated for each one of them using the Monte Carlo simulation method. Based on internal delays calculated samples, distribution functions for each element are calculated, which can be used to increase the accuracy of the proposed model. The proposed method in this paper helps designers model high-level elastic circuits using xMAS and validate their timing and functionality results. Since there are different implementations with specific properties for each elastic element, there will be a wide range of choices for each design. Therefore, designers can evaluate a wide range of possible designs in a short period of time.

The main contributions of this paper are as follows:

- A scalable high-level modeling platform is introduced, which could be used to model complex elastic architectures. The designer can explore and compare different architectures of an elastic circuit and verify their correctness.
- Timing verification of elastic circuits is available using the mentioned platform while process variations are included.
- The Initial delay of all the components of elastic circuits has been modeled with high accuracy, and used in their timing verification.

The rest of this paper is as follows: In Section 2, elastic circuits, handshaking protocols, and their elements are introduced. xMAS and SAN, two modeling formalisms that are used in this paper, are also introduced in this section. Suggested modeling and verification methods can be found in Section 3. The result on different elastic elements' functionality in the presence of variation, functional and timing verification of elastic circuits are discussed in Section 4. Finally, the paper ends with conclusion, available research opportunities and remaining challenges in this field.

2 PRELIMINARY

2.1 Elastic Circuits

The central concept in Elastic circuits is to use handshake protocol signals between predecessors and successors to implement circuits with latency-insensitivity capability. A *valid* signal indicates the validity of the data that the predecessor(s) send to the successor(s), whereas in the opposite direction, the *stop* signal informs the predecessor(s) that a component cannot accept new data. In synchronous elastic circuits, as in synchronous circuits, events occur in the presence of a clock.

The technique that converts ordinary clock circuit into an elastic design is called synchronous elasticization. Figure 1 shows an example of this technique. Figure1.a pictures a synchronous circuit consisting of registers and combinational logic (CL). A common method to elasticize a circuit is achieved by replacing each flip-flop in the original synchronous system with elastic buffers that have their own control section. Each register-to-register data communication requires an elastic control channel to control the data flow between the two registers. Based on the SELF protocol [22], a control channel is composed of *valid* and *stop* signals. In a network of elastic control channels, the fork and join components are used as shown in Figure 1.b via Ⓕ and Ⓙ symbols. Various designs have been proposed for elastic buffer, fork, join and channel protocols [8, 16, 23, 24]. Choosing among any of these elements for a design comes with its own advantages and disadvantages, such as differences in complexity, performance, power, variation tolerance, and error.

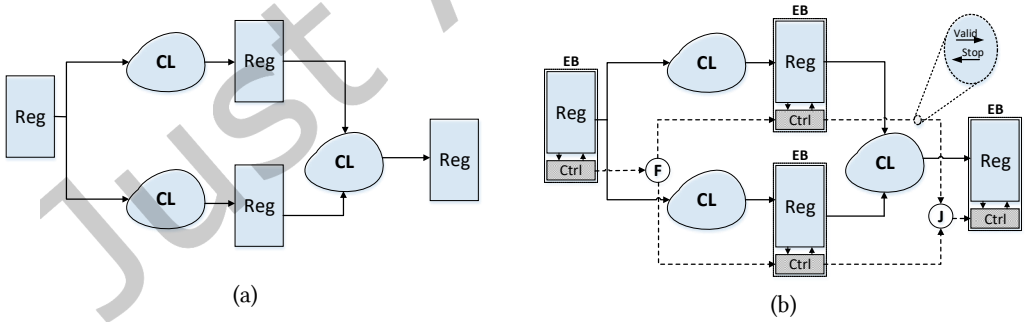


Figure 1: Synchronous elasticization (a) Synchronous version (b) Synchronous elastic version

2.1.1 *Handshaking protocols*

Handshaking protocols in Elastic circuits can be classified into two main protocols; push- and pull-based. In a push-based protocol, the predecessor sends the data along with the *valid* signal as long as the data is

available and the successor has not yet asserted the *stop* signal. In pull-based protocols, the successor first declares its readiness to receive data then the predecessor sends the data along with the *valid* signal.

In [25] authors proposed a protocol that is categorized into pull-based protocols. In this protocol, *valid* and *ready* control signals are used. The successor asserts *ready* signal when it is available for receiving data, then the predecessor sends the new data along with the *valid* signal. In this protocol, if the successor deasserts the *ready* signal, the predecessor stops data transmission.

The proposed protocols in [22, 26, 27] are examples of studies in the push-based protocols category. The choice of a protocol has a direct effect on the structure of the circuit components. In [16] authors show that the proposed protocol in [22], called SELF (the protocol used in this paper), functions better than other protocols in terms of performance and power consumption.

As shown in Figure 2, the SELF protocol consists of three states. When the predecessor has no data to send, the SELF protocol goes into Idle state. If the predecessor wants to send data and the successor is ready to receive data, the protocol is in Transfer state and if the successor is not ready to receive data, it goes to Retry state.

The control network in elastic circuits, which is responsible for carrying valid and stop signals, consists of buffer, fork and join components.

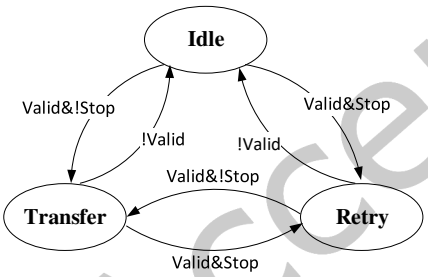
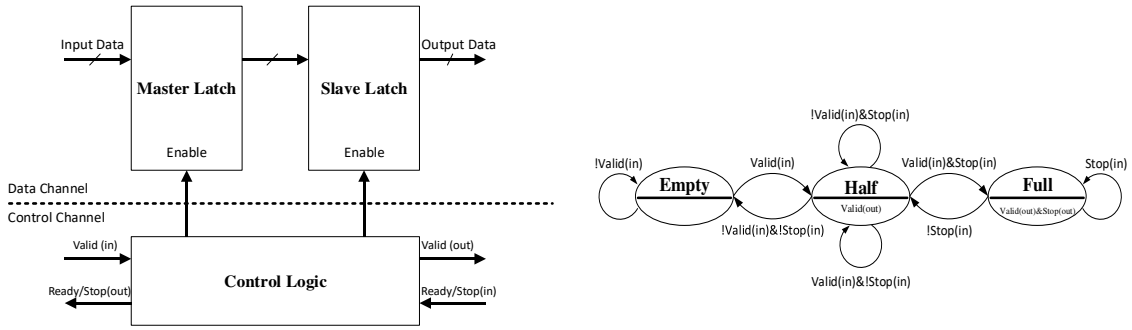


Figure 2: SELF protocol state diagram

2.1.2 Elastic components

2.1.2.1 Elastic buffers

By replacing all flip-flops in the synchronous system with elastic buffers, in addition to pipelining feature, a synchronous elastic system allows the clock pipeline to be stalled. Figure 3.a shows the concept of an elastic buffer. The elastic buffer is the elementary storage structure in elastic circuits and consists of data and control logic. The control logic design based on SELF protocol can be found in [24]. This elastic buffer design could be divided into two independent parts; each part is called elastic half buffer (EHB). Depending on how many data tokens are there in the data of an elastic buffer, it can be in one of three states: Empty, Half, or Full (Figure 3.b).



(a) (b)

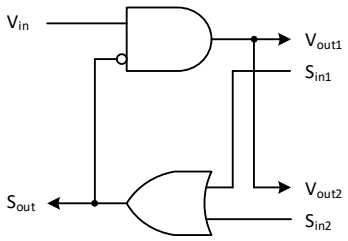
Figure 3. Elastic buffer (a) components and (b) state diagram

2.1.2.2 Lazy forks

The lazy fork propagates valid data received at the input to its outputs; it consigns data only when all successors are ready. According to this definition, the truth table can be obtained as shown in Figure 4.a. Based on SELF protocol, if a *stop* signal (S_{in}) is observed at an output and a *valid* signal (V_{in}) is asserted at the input, the *valid* signal of that output can be "Don't care", because in this case, the channel cannot be in Transfer state. In the presented lazy fork truth table (Figure 4.a), V_{out1} results are shown based on V_{in}, S_{in1} and S_{in2}. V_{out1} is zero when V_{in}=0, so no *valid* signal should be asserted at the output. V_{out1} will be equal to 1 and Transfer state will occur while V_{in}=1, and no *stop* signals are received from neither S_{in1} nor S_{in2}. On the other hand, V_{out1} must be zero while S_{in2}=1, S_{in1}=0, and V_{in}=1, so the channel will not turn into Transfer state. But when V_{in}=1 and S_{in1}=1, V_{out1} could be either 0 or 1, regardless of the value in S_{in2}, since the channel won't turn into Transfer state in either case. X_1 and X_0 could be defined for V_{out2} via the same method. Therefore, four different implementations for lazy fork can be provided by considering "Don't care" signals. Each implementation is indicated by LF_{X1X0} (e.g. LF_{00} means that X_1 and X_0 are both considered zero in the implementation). Each of the LFs can have different characteristics and is prone to errors in elastic circuits that need to be investigated.

V_{in}	S_{in1}	S_{in2}	V_{out1}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	X_0
1	1	1	X_1

(a)



(b)

Figure 4: (a) Truth table of lazy fork and (b) LF00 implementation

2.1.2.3 Eager fork

Similar to lazy fork, eager fork (EF) replicates valid data received at the inputs to its outputs, but even when not all its outputs are ready to receive, it forwards valid data independently to each output that is ready. The eager fork does not accept any new valid data until all successors have consumed the previous data. Implementing eager fork requires two flip-flops to remember which successor already received the data. The structure is presented in [24].

2.1.2.4 Lazy joins

A join waits to receive valid data on all of its inputs before transferring the valid data on its output. The lazy join truth table is shown in Figure 5. According to the four "Don't care" signals, there are 16 different implementations for lazy join. Similar to lazy fork, we specify each lazy join as $LJ_{X3X2X1X0}$.

S_{in}	V_{in1}	V_{in2}	S_{out1}
0	0	0	X_0
0	0	1	X_1
0	1	0	1
0	1	1	0
1	0	0	X_2
1	0	1	X_3
1	1	0	1
1	1	1	1

Figure 5: Truth table of lazy join

2.2 xMAS modeling

xMAS[17] is a high-level modeling formalism that demonstrates significant improvement in the modeling of communication systems. The xMAS model consists of a set of primitives that are connected by channels. Each channel comprises three signals: *irdy*, *trdy* and *data*. *irdy* indicates the validity of the data while both signals are controlled by the initiator primitive. *trdy* indicates the readiness of the target primitive to receive data. xMAS primitives are depicted in Figure 6.

Source and the sink primitives input or output data in the form of packets or tokens as the xMAS model interfaces with the environment. Fork and join are used as synchronizers in the model. A fork transmits a token from input to outputs when all outputs are available for receiving a token, and there is a token at the

input to be sent. Fork's inverse operation is accomplished via Join, which transmits tokens when there is a token in all inputs, and the output is ready to receive. A function primitive transforms input data to output by using a deterministic function. The switch primitive is used for routing tokens through the model and depending on whether a condition on the input is true or false, the token is routed to one of the outputs. Merge primitive (a.k.a arbitration primitive) selects a token among one of the inputs and produces it on the output. Queue primitive consists of several slots that hold data or tokens. The data is read from the head slot and written to the tail slot. The capacity is represented by a non-negative integer such as "*k*". This paper briefly describes the xMAS components. Further explanations on the subject can be found in [17] and [28].

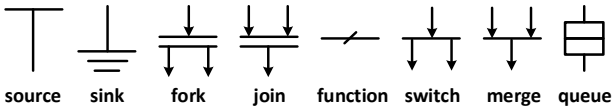


Figure 6: xMAS primitives

2.3 SAN Formalism

Deterministic and statistical timings, probability and distribution functions can be modeled by using Stochastic Activity Networks (SANs). These formalisms are stochastic extensions to Petri nets [29]. The SAN model can be used for various purposes, such as evaluating the performance, verifying, and analyzing the reliability. SANs consist of four primitives as listed below:

- Places: a place holds a token to represent the state of the model system.
- Activities: actions in the model system that take a specified amount of time to complete are known as activities. There are two types of activities: instantaneous and timed. Each timed activity has deterministic timing or statistical timing based on a specific distribution function associated with its duration.
- Input gates: enabling activities are controlled by input gates. Input gates can be defined by enabling predicate and a Boolean function that controls whether the connected activity is enabled.
- Output gates: Output gates define the marking changes. These changes take place when the activity is completed via a function.

We have used Mobius toolset [30] to simulate SAN models. It can define a variety of well-known distribution functions such as Exponential, Normal, and Log-normal. Iterations for simulations can be of any desired number in this toolset. Finally we can analyze the model using a trace file generated in each simulation.

3 DESCRIPTION OF THE MODELING METHOD

Figure 7 shows different steps of our proposed method to model and verify elastic circuits. In the first step, each type of elastic elements is modeled at high-level using xMAS. We have used the Workcraft tool [31, 32] to construct the xMAS model. This tool contains an xMAS plug-in allowing us to build graphical xMAS models. The Workcraft tool allows the xMAS components to be organized into a distinct group thanks to its tool control panel, so each of these xMAS models of elastic elements can be placed in a module.

Therefore, complex elastic circuits can be modeled easily. Workcraft supports xMAS simulation and analysis by converting it to STGs and utilizing established verification functionality. In addition to the existing verification features in Workcraft, we have added accurate timing analysis by considering process variation in our flow verification. We implemented the SAN model translator alongside the Workcraft tool that we used, which translates xMAS model representations into SAN representations.

Each elastic component's internal delay model is extracted via HSPICE Monte Carlo simulation by considering process variation and embedded in activity delays of the SAN model. For timing verification, we have used the Mobius tool, which can receive timing characteristics. Finally, elastic circuits verification with delay variability is examined.

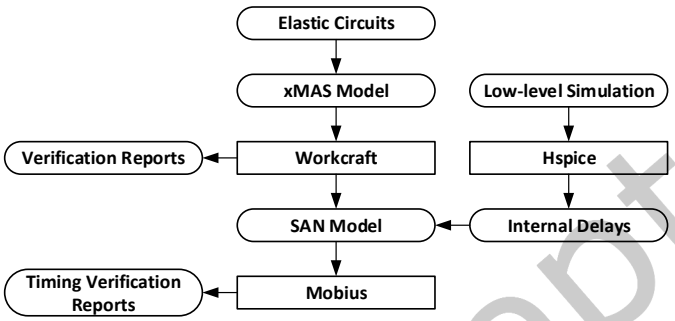


Figure 7: Modeling and Verification Flow

3.1 xMAS Modeling of Elastic components

Buffer, fork, and join are used in elastic circuits' control network. There are various designs and implementations for each of these elements (i.e., a fork can have five different implementations: EF, LF_{00}, LF_{01}, LF_{10} and LF_{11}). Selecting any combination of these elements can result in a design with different characteristics. All possible combinations of fork and join implementations, as well as the elastic buffer mentioned in this paper, which could be found in [24], are modeled by xMAS. One sample of the xMAS model in each type of elastic components (model of LF_{00} in forks, LJ_{0000} in joins, and EHB in buffers) are provided in Figure 8. Elastic buffer can be built by connecting two EHBs. The LF and LJ xMAS models use source, sink, fork, join and merge primitives. LF_{00} consists of an input channel and two output channels. Input channels consist of V_{in} and S_{out} signals. V_{out1} and S_{in1} are the signals of one of the output channels, while the other channel's signals are S_{in2} and V_{out2}. Based on the truth table of LF_{00}, V_{out1} and V_{out2} generate a *valid* signal when S_{in1}=0, S_{in2}=0 and V_{in}=1. In configured fairness algorithms for merge primitives, different ports are prioritized. A port with a lower priority is chosen when a port with a higher priority has no tokens to transfer. Here, S_{in1} and S_{in2} signals are connected to Merge 1 and Merge 2 ports with higher priorities. As a result, in the lack of a signal from S_{in1}, a token is generated from Source 1 and Fork 5 paths. If both Join 2 inputs include a token (meaning that there is no *stop* signal in input channels) a token is sent to Fork 3. the primitive of Fork 3 sends the incoming token to a Join 1 via an output channel. Join 1 uses the token in V_{in} (in case there is one) until a token is delivered to V_{out1} and V_{out2} from Fork 2. For S_{out}, if (S_{in2} OR S_{in1}) = 1, *stop* signal is generated via Source 3, Sink 2, Merge 3 and Fork 4 primitives. In addition to the mentioned primitives for LF and LJ, the queue is also used in EHB and EF models. The

queue is needed to keep the status of the EHB and two queues are required in EF to recall which output channels already received the data. Due to the modularity of the Workcraft, each elastic element in the xMAS model can be placed in a module and by connecting modules together, greater models can be built.

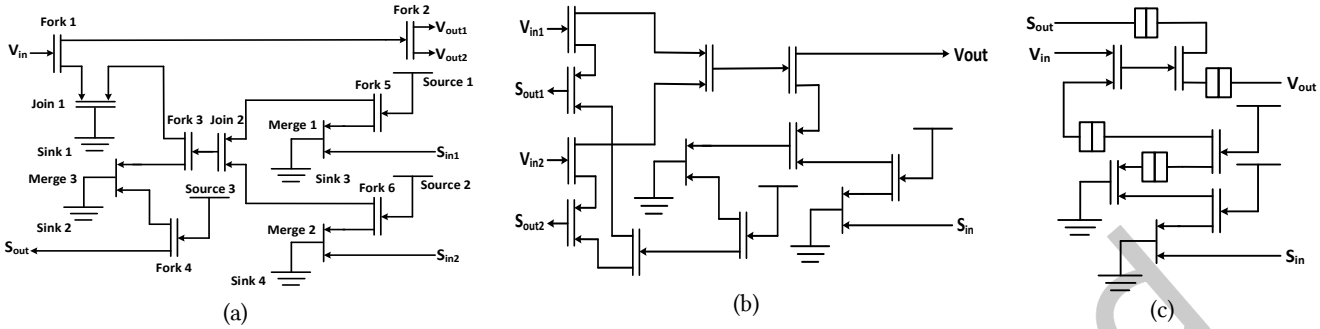


Figure 8: xMAS model of elastic components (a) LF_{00} (b) LJ_{0000} (c) EHB

3.2 xMAS to SAN Translator

We translate the xMAS model to the SAN model by implementing a translator within our Workcraft. The translator accepts an xMAS model of elastic circuits in data interchange format (JSON) and translates it into the SAN model. As a result, by embedding the delay distribution into the model, the timing verification analysis of elastic circuits becomes affordable by using the Mobius toolset. For this purpose, each of the xMAS primitives needs to be represented in their SAN model and eventually connected to each other to make more complex models. The following diagrams provide the translation for some of the key primitives.

Figure 9 shows the translation of the xMAS source primitive. The source primitive is used to model the creation and injection of tokens in the model, which is parameterized by a constant expression $e : \alpha$. The source has a single output port $o : \alpha$ and signal relations are formally defined by the following equations:

$$o.irdy := \textbf{\textit{oracle or pre}}\ (o.irdy\ and\ not\ o.trdy) \quad (1)$$

$$o.data = e \quad (2)$$

In each complete cycle, the source non-deterministically tries to send a packet *e*. *pre* in Equation 1 represents the standard synchronous operator, which returns zero in the first cycle and the value of its (Boolean) argument in preceding cycle; and the non-determinism of the source is modeled using an *oracle*. The $o.trdy$ in Equation 1 defines the source's persistency, regardless of *oracle*.

In the SAN model of the source, we consider three places for output signal ports (o.trdy, o.irdy, and o.data) and three places for input signals. Since in each cycle it is necessary to know the status of *o.irdy* and *o.trdy* in the previous cycle for *pre*, clk_time activity is used based on the global clock. We have used producer and oracle places and an activity called "produce_distribution" in the SAN model to model oracle; produce_distribution can accept random variables or use arbitrary distributions to generate tokens. IG1 is an input gate, which is configured to select pre or oracle for o.irdy and o.data.

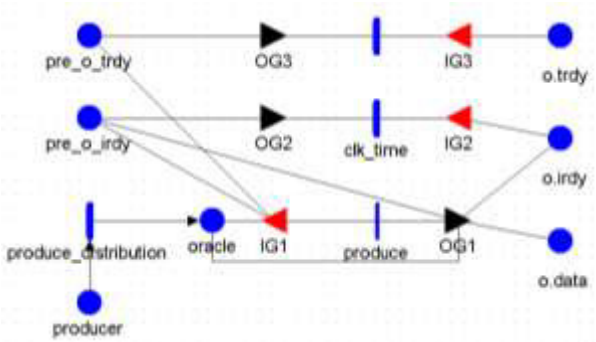


Figure 9: SAN translation-source

Sink primitive is the dual component of source which consumes a token through one input port, *i*, as described in equation 3:

$$i.trdy \coloneqq \boldsymbol{oracle\ or\ pre}(i.trdy\ and\ not\ i.irdy) \quad (3)$$

In Figure 10, the sink translation process to the SAN model is shown. It is similar to the source translation model, with one difference; here, the sink consumes tokens in oracle or pre.

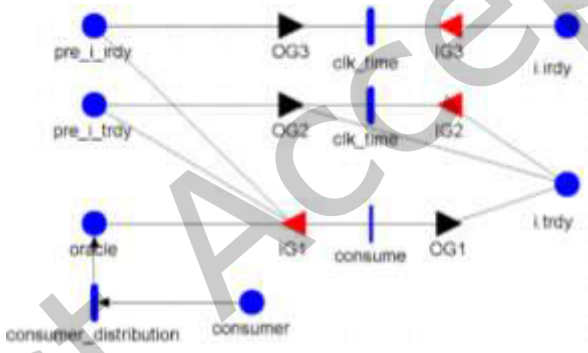


Figure 10: SAN translation-sink

Figure 11 shows the translation process of xMAS fork primitive. The fork primitive has one input port, *i*, and two output ports, *a* and *b*. By definition, the fork signal relations can be formally defined using the following equations:

$$a.irdy \coloneqq i.irdy\ and\ b.trdy \quad (4) \qquad a.data \coloneqq f(i.data) \quad (5)$$

$$b.irdy \coloneqq i.irdy\ and\ a.trdy \quad (6) \qquad b.data \coloneqq g(i.data) \quad (7)$$

$$i.trdy \coloneqq a.trdy\ and\ b.trdy \quad (8)$$

Instantaneous activity is used in the SAN model of fork, for instance, token transfer from input to output. Each input gate is configured to accept its input signals and produce an output signal.

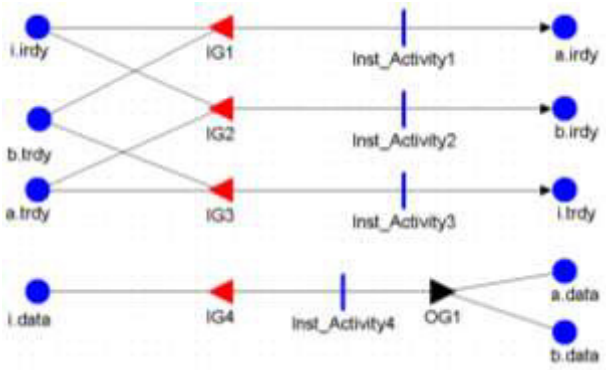


Figure 11: SAN translation-fork

As the dual component of a fork, Join takes two inputs, *a* and *b*, and produces one output, *o*. The formal definition of Join is as follows:

$$a.trdy \coloneqq o.trdy \; and \; b.irdy \quad (9) \qquad b.trdy \coloneqq o.trdy \; and \; a.irdy \quad (10)$$

$$o.irdy \coloneqq a.irdy \; and \; b.irdy \quad (11) \qquad o.data \coloneqq h(a.data, b.data) \quad (12)$$

Figure 12 shows the translation process of join primitive to SAN. It consists of five places for input signals and five places for output signals, and similar to the fork, consist of input gates that is configured to produce output signals.

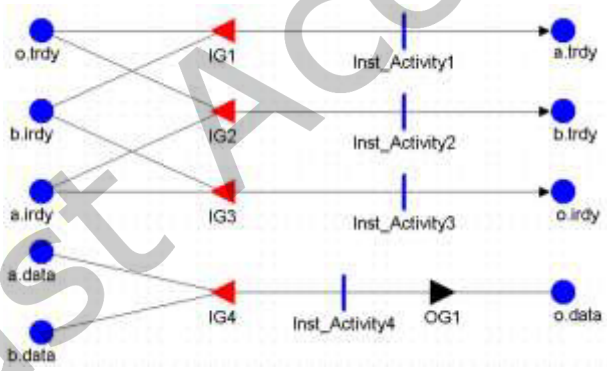


Figure 12: SAN translation-join

Merge primitive (consisting of two input ports, *a* and *b*, and one output port, *o*) is used for selecting token through its inputs. Formally it is defined as the following equations:

$$o.irdy \coloneqq a.irdy \; \mathbf{or} \; b.irdy \quad (13)$$

$$o.data \coloneqq a.data \; \mathbf{if \; not} \; u \; \mathbf{and} \; a.irdy \quad (14)$$
$$b.data \; \mathbf{if} \; u \; \mathbf{and} \; b.irdy$$

$$b.trdy \coloneqq u \ \textbf{and}\ o.trdy \ \textbf{and}\ b.irdy \quad (15)$$

$$a.trdy \coloneqq \textbf{not}\ u \ \textbf{and}\ o.trdy \ \textbf{and}\ a.irdy \quad (16)$$

Local Boolean state variable, u is used to ensure fairness. In our model, we have defined the fairness algorithm as follows:

$$u \coloneqq \begin{matrix} 1 & \textbf{if}\ b.irdy \\ 0 & \textbf{if not}\ b.irdy \ \textbf{and}\ a.irdy \end{matrix} \quad (17)$$

This equation explains that b has higher precedence than a, which means port a is selected, only if b is not ready. The fairness algorithm can be configured in input gates of the SAN model. The SAN model of merge primitive is shown in Figure 13.

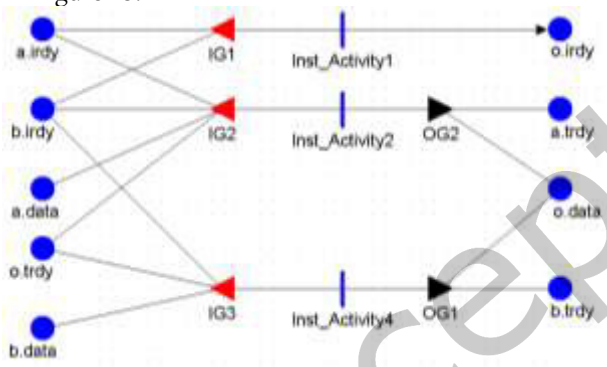


Figure 13: SAN translation-merge

3.3 Internal delay distribution extraction

The impact of variations of the process will substantially grow with transistors' size shrinkage and can affect the functionality and behavior of a circuit or a system [33]. Random and systematic variations are two types of process variation impacts. In [15] detailed examples on systematic variations effects on adjacent transistors are studied. As an example of systematic variations, some sources such as lithographic and etching processes have a similar effect on adjacent transistors, whereas some sources of variations like voltage threshold cause random effects on adjacent transistors[15]. Statistical methods are used to extract the internal delay of elastic circuit primitives under process variation. The random effect of variation parameters is studied using HSPICE Monte Carlo simulation. The effects of voltage threshold as an important source of process variation were investigated in this paper. Although other forms of process variation sources are not covered in this paper, our proposed approaches can be used to analyze them.

Coefficient of variation (C_v) is used to analyze and compare the variation effects on elastic primitives. It depicts how delay or power changes based on variation [34-36]. Following equations are used to calculate coefficient of variation:

$$\mu(x) = \frac{\sum_{i=1}^{n} x_i}{n} \quad (18) \ , \ var(x) = \frac{\sum_{i=1}^{n} (x_i - \mu(x))^2}{n-1} \quad (19) \ , \sigma(x) = \sqrt{var(x)} \quad (20) \ , C_v = \frac{\sigma(x)}{\mu(x)} \quad (21)$$

The fittest distribution function for delay samples is obtained by using the Maximum Likelihood Estimation (MLE)[37] approach. For each elastic component, the probability based on distribution function is embedded in its SAN model.

In various references, Normal distribution function is used to model delay[35]. It is observed that the Normal distribution function cannot properly define circuits' delay distribution when simulations are based on asynchronous pipeline circuit variations, therefore we have made use of the Log-normal distribution function for that propose. We will also show that with the Log-normal distribution function, more accurate results for elastic circuits can be obtained.

3.4 Verification

Functional and timing verifications are two available verification approaches for elastic circuits. All verifications are based on the SELF protocol is used as the handshaking protocol for elastic circuits in this paper. Functional verification is performed based on xMAS and Workcraft tools in which elastic circuits are investigated in terms of functionality. In the first step, all elastic components are tested for persistency, liveness, and absence of deadlocks. In the next step, different scenarios on connections between elastic components to one another and elastic control networks are tested. Tested properties are listed below:

- Persistence: transition from Retry to Idle should not happen in any channel. This property is tested via persistence.
- Deadlock freedom: absence of deadlock at each elastic component means there are at least two available states from each reachable state.
- Liveness: to verify this property, it must be proved that transferred tokens in all channels agree in Transfer mode.

In timing verification which is, to the best of our knowledge, performed for the first time in this paper, elastic functionalities are tested by considering variations and elastic internal delays. In this verification, data channel and control channel synchronizations are tested under variations. One of the properties that can be tested in this area is

$$t_{valid(i-j)} + t_{enable} \geq t_{data(i-j)} + t_{setup} \quad (22)$$

where $t_{valid(i-j)}$ represents *valid* signal's transition from *i* to *j* in the control path. If there are elastic components such as join and fork in the path as well, their respective delays must be included. t_{enable} represents the latch activation time, and the setup time for each latch is represented by t_{setup}, which is a technology-dependent parameter. Finally, $t_{data(i-j)}$ shows the transition delay from *i* to *j*. This equation tells us that to latch data correctly, the latch enabling delay must be bigger than the setup time. Data transition times and *valid* signals are presented in Figure 14.

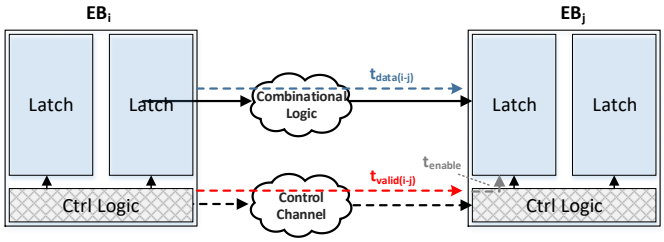


Figure 14: Timing Verification scenario

4 EXPERIMENTAL RESULTS

4.1 Performance evaluation and variation study

To analyze the effect of variations on elastic components, Monte Carlo simulation (with 256 iterations) is performed on each one of them. In Figure 15, the considered scenarios are presented to simulate join and fork. The simulations were performed based on 32nm technology. Other simulation's parameters are listed in Table 1. Initially, the average values for delay, Power-Delay-Product (PDP) and power have been calculated for each component by considering voltage threshold (V_{th}) variation of 20%, 25%, 30%, and 35%; then, the variation effects have been calculated on each component based on Equations 18 to 21.

Table 1: Simulation parameters

Parameters	Values
V_{dd}	0.9 V
V_{th}	0.16 V
T_{ox}	1 nm
L_{eff}	12.6 nm
L_n=L_p	32 nm
Wn=Wp/2	128 nm
Process	Typical-Typical
Temperature	25°C
Simulation Length	10 ns
Distribution	Gaussian

Average values for delay, PDP, and power with considering voltage threshold variation for different forms of elastic forks are presented in Figures 16 to 18. As seen, the increase in variation affects more, power than delay. Due to the complexities of EF circuits (in comparison with LF circuits), the average power for LF_{00} is at its minimum value as expected. While the average power is maximum for EF, LF_{00} and LF_{10} show smaller delays than other forks. Trends show that LF_{00} and LF_{10} have lower PDP values with respect to other forks while EF has greater PDP values in comparison with others. As seen in Figure 19, coefficient of variation (C_v) parameter is used to show the variation effect on different elastic fork implementations. LF_{10} is the least affected implementation by variation while LF_{11} is the most affected.

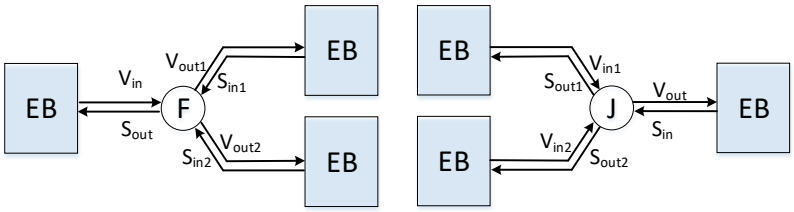


Figure 15: Variation effects simulation scenarios

Performance analysis under variation is also performed on different implementations of join. The average value for power, delay and PDP components of join for 20%, 25%, 30% and 35% of V_{th} variation are presented in Figures 20 to 22. Investigations show that LJ_{1111} represents lower power in comparison with other joins while LJ_{1100} and LJ_{0111} represent the largest values in terms of delay. Finally, LJ_{1111} has the lowest PDP values in all variations. C_v values in Figure 23 indicate that LJ_{1111} is the least affected join implementation in the presence of variations compared to other join implementations.

In [16] different elastic buffers are tested for variations. The results show the presented elastic buffer in [24] has the best performance in the presence of variations. We have used the presented elastic buffer in [24] for our scenarios.

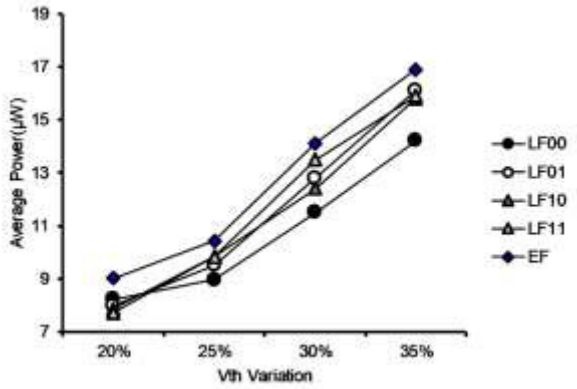


Figure 16: Average power for elastic fork with V_{th} variation consideration

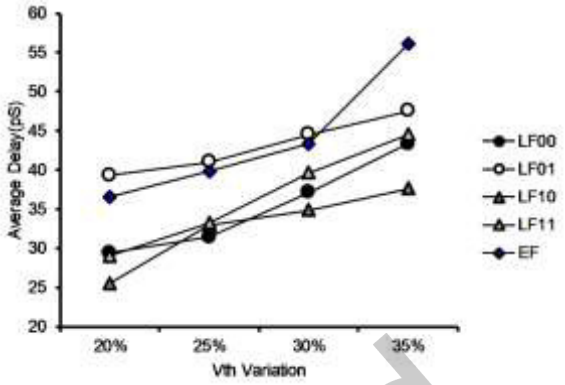


Figure 17: Average delay for elastic fork with V_{th} variation consideration

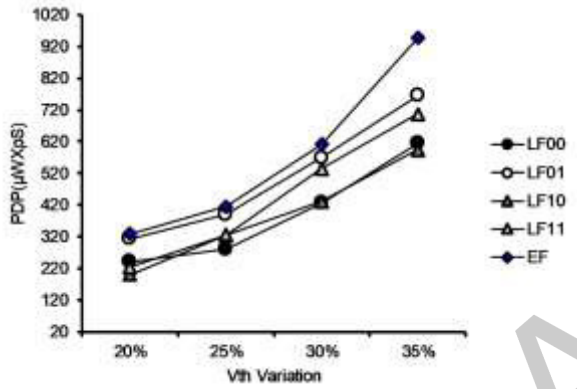


Figure 18: Average PDP for elastic fork with V_{th} variation consideration

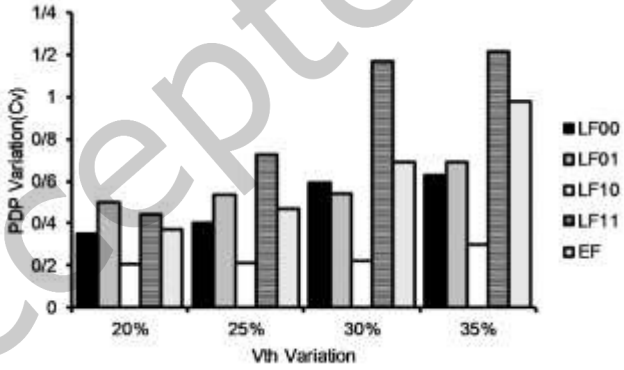


Figure 19: Variation effects on various forms of elastic fork

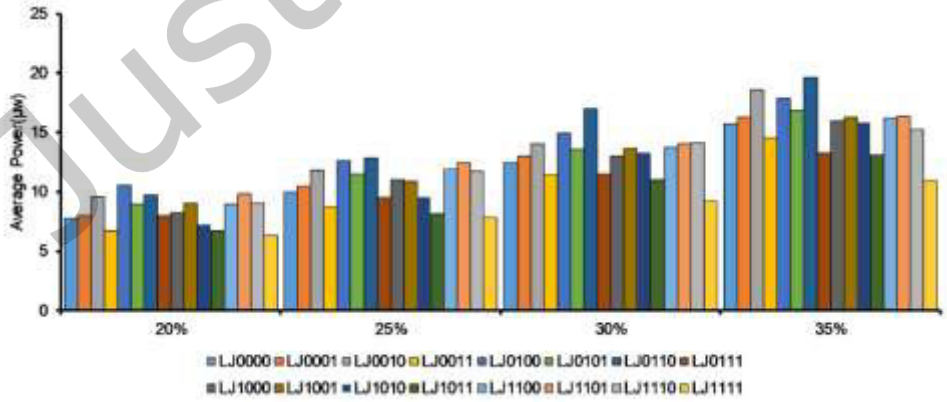


Figure 20: Average power for elastic join with V_{th} variation consideration

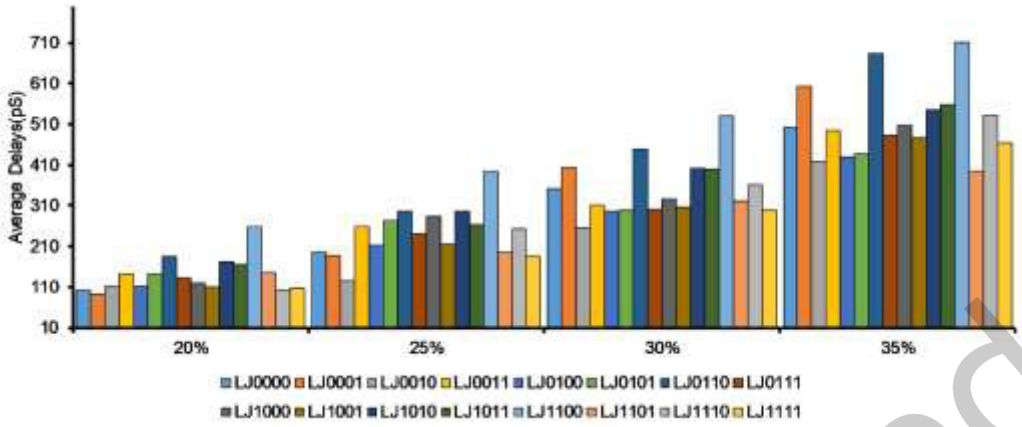


Figure 21: Average delay for elastic join with V_{th} variation consideration

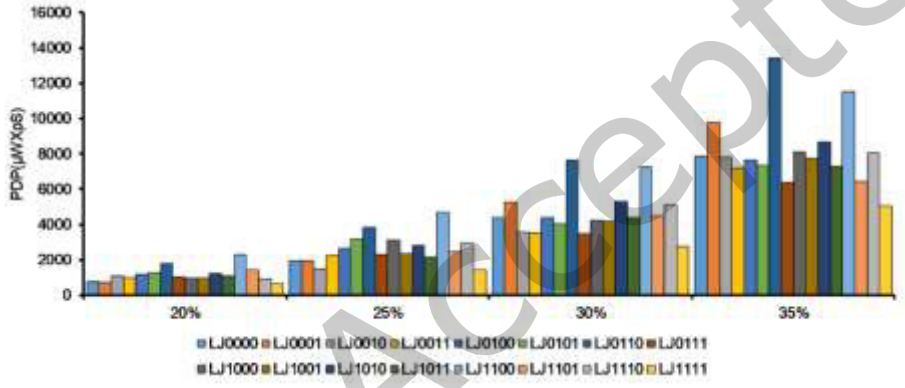


Figure 22: Average PDP for elastic join with V_{th} variation consideration

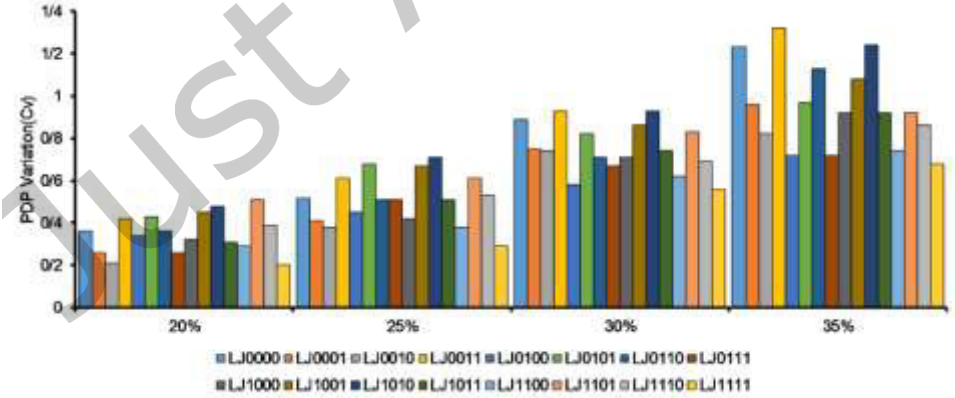


Figure 23: Variation effects on various forms of elastic join

4.2 Delay distribution extraction

MLE parameter is used to choose the best internal delay distributions of elastic components. Different distributions can be fitted on delay samples, and the respective distribution parameters are calculated precisely by using this parameter. In other words, a distribution is best fitted if its respective MLE value is larger. Normal distribution PDFs (probability distribution functions) and Log-normal distributions are observable alongside with internal delays of LF_{00} (delay: $t_{[Vin+\rightarrow Vout1+]}$), LJ_{0000} (delay: $t_{[Vin1+\rightarrow Vout+]}$) and EB (delay: $t_{[Vin+\rightarrow enable+]}$) in Figures 24 to 26, where 'enable' represents Master Latch activation signal in EB. The Log-normal distribution shows higher values in comparison with Normal distributions in each figure.

Calculated MLE values for Normal and Log-normal distributions are presented in Table 2. As seen in this table, all join implementations are fitted by the Log-normal distribution function except for two of them. Normal and Log-normal distribution function equations are presented below:

$$\text{Gaussian PDF: } f(x)=\frac{1}{\sigma\sqrt{2\Pi}}e^{\frac{-(x-\mu)^2}{2\sigma^2}}\left(X\sim N(\mu,\sigma)\right) \quad (23)$$

$$\text{Log-normal PDF: if } X\approx N(\mu,\sigma)\text{, Then } Y \text{ is Log-normal if } y=e^x,\ Y\sim LnN(N(\mu,\sigma)),f_y(y)=\frac{1}{y\sigma\sqrt{2\Pi}}e^{\frac{-(\ln y-\mu)^2}{2\sigma^2}} \quad (24)$$

Finally, each calculated distribution function is placed in elastic components of the SAN model using activity elements.

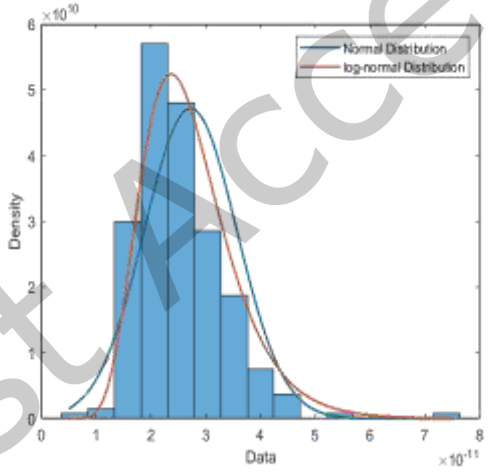


Figure 24: LF_{00} distributions

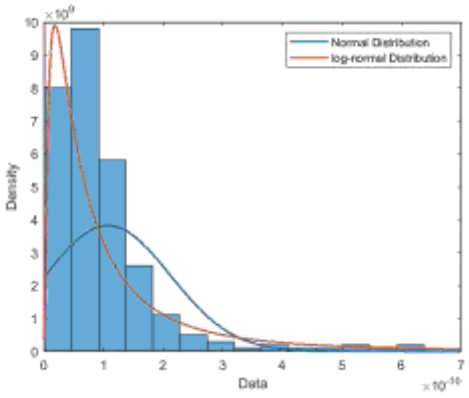


Figure 25: LJ_{0000} distributions

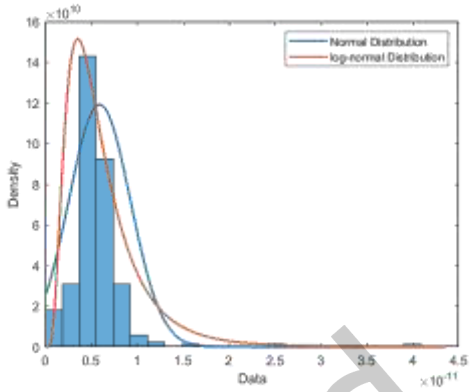


Figure 26: Elastic buffer distributions

Table 2: MLE value for elastic components distributions

Elastic Component	Normal MLE	Log-normal MLE	BestFitDistribution
LF_{00}	2765.7	3085.7	Log-normal
LF_{01}	2776.9	2804.1	Log-normal
LF_{10}	2819.4	2873.8	Log-normal
LF_{11}	2948.8	3080.0	Log-normal
EF	6215.3	6723.5	Log-normal
LJ_{0000}	3785.6	3872.0	Log-normal
LJ_{0001}	3205.1	3203.9	Normal
LJ_{0010}	4023.3	4157.8	Log-normal
LJ_{0011}	3684.9	3824.2	Log-normal
LJ_{0100}	2999.7	3278.9	Log-normal
LJ_{0101}	3478.1	3614.7	Log-normal
LJ_{0110}	3222.7	3189.5	Normal
LJ_{0111}	3758.6	3759.8	Log-normal
LJ_{1000}	2641.2	2836.4	Log-normal
LJ_{1001}	4033.7	4102.3	Log-normal
LJ_{1010}	3412.3	3652.9	Log-normal
LJ_{1011}	3225.6	3275.5	Log-normal
LJ_{1100}	3014.7	3018.1	Log-normal
LJ_{1101}	3625.3	3696.8	Log-normal
LJ_{1110}	2910.4	3129.8	Log-normal
LJ_{1111}	3666.6	3722.7	Log-normal
EB	5837.4	5848.7	Log-normal

4.3 Verification Results

4.3.1 Functional verification

To verify the correctness of each elastic component, we have used the presented scenarios depicted in Figure 15. Verifications are performed via the Workcraft tool and trace file analysis. To verify persistency, each channel of each component needs to be checked to verify that there is no $Retry \rightarrow Idle$, which is done by analyzing the channel's *valid* and *stop* signals, so that transitions from $valid \wedge stop$ to $!valid$ never takes place. To verify the absence of deadlocks for elastic components, we have used the existing properties in Workcraft, and to verify components in terms of liveness, the transited tokens in each channel need to be calculated and compared. This is accomplished by using simulation traces in Workcraft. Finally, the verification status of each component for the mentioned properties is listed in Table 3. In this table, elastic components which satisfy the corresponding requirements have been indicated by ✓, while the ones that were unable to satisfy the properties have been marked by ×.

Based on the performed verifications, all elements include deadlock freedom and liveness properties. The results show that three forks (from a total of 5 forks), LF_{00}, LF_{10} and EF, have performed correctly in terms of persistency and 6 joins from 16 existing forks have shown persistency properties. For example, LF_{01} violates persistency properties, as its V_{out1} output is derived from the following equation (based on truth table in Figure 4.a):

$$V_{out1}=V_{in} \cdot S'_{in2} \quad (25)$$

For persistency properties, transfer from Retry to Idle state must be locked. Though as could be comprehended from Equation 25, V_{out1}=1, while V_{in}=1, S_{in1}=1 and S_{in2}=0, and the channel is in Retry state, but when S_{in2} =1 then V_{out1} = 0, and the channel will go to Idle state, it can be concluded that LF_{01} does not maintain persistency properties.

These ten elements that passed the functional verification properties will be tested for timing verification in the next section, while those which could not satisfy the functional verification requirements will be excluded.

Table 3: Functional verification status results

Elastic Component	Persistenc y	Deadlock freedom	Liveness
LF_{00}	✓	✓	✓
LF_{01}	×	✓	✓
LF_{10}	✓	✓	✓
LF_{11}	×	✓	✓
EF	✓	✓	✓
LJ_{0000}	✓	✓	✓
LJ_{0001}	×	✓	✓
LJ_{0010}	✓	✓	✓
LJ_{0011}	✓	✓	✓
LJ_{0100}	×	✓	✓
LJ_{0101}	×	✓	✓
LJ_{0110}	×	✓	✓
LJ_{0111}	×	✓	✓

LJ_{1000}	×	✓	✓
LJ_{1001}	×	✓	✓
LJ_{1010}	✓	✓	✓
LJ_{1011}	✓	✓	✓
LJ_{1100}	×	✓	✓
LJ_{1101}	×	✓	✓
LJ_{1110}	×	✓	✓
LJ_{1111}	✓	✓	✓
EB	✓	✓	✓

4.3.2 *Timing Verification*

For time verifications of elastic circuits, we have used a scenario described in Figure 14, in which two EBs are connected. In different scenarios different fork and join elements are used and the results are calculated for different values of variation (20%, 25%, 30%, 35%) via the following equation:

$$t_{valid(i-j)} + t_{enable} \geq t_{data(i-j)} + t_{setup} \quad (26)$$

Time verification results are reported in Table 4. Each scenario represents the placement of an elastic component in the control channel by considering 20%, 25%, 30% and 35% values for V_{th}. As an example, LF_{00} means an LF_{00} implementation is used in the control channel (it is the same for other forks and joins). In the EB scenario, the controls of two EBs are connected directly. Each scenario is repeated 1024 times, and the presented values show the number of equation's correctness validations with regard to the number of total iterations. Presented results in this table indicate that variation affects elastic circuits' functionality; the probability of an elastic circuit's malfunction directly depends on the magnitude of variation. For example, there is 0.59% chance of error for LF_{00} while V_{th}=35% which leads to a malfunctionality in synchronization between control channel and data channel. It can be concluded that EF and LF_{10} among fork elements and LJ_{0010} and LJ_{1111} among join elements are less likely to be subject of error.

Table 4: Timing verification results

Elastic Component	V_{th} = 20%	V_{th} = 25%	V_{th} = 30%	V_{th} = 35%
LF_{00}	0.9990	0.9980	0.9960	0.9941
LF_{10}	1	1	0.9980	0.9960
EF	1	1	1	0.9990
LJ_{0000}	1	0.9990	0.9990	0.9931
LJ_{0010}	1	1	0.9990	0.9990
LJ_{0011}	0.9980	0.9980	0.9951	0.9912
LJ_{1010}	1	0.9990	0.9990	0.9980
LJ_{1011}	1	1	0.9990	0.9960
LJ_{1111}	1	1	0.9990	0.9990
EB	0.9892	0.9775	0.9775	0.9765

4.3.3 *Functional and timing verification of MiniMIPS elastic processor*

In this section, the MiniMIPS microprocessor introduced in [38] is used to evaluate the proposed method. This microprocessor is an 8-bit subset of MIPS microprocessor architecture [39, 40], which is widely used

in different works, and is so comprehensive that can show all the properties of the method introduced in this paper. A thorough introduction to this microprocessor could be found in [38].

Initially, our MiniMIPS synchronous architecture must be elasticized. For this purpose, all registers must be replaced with Elastic Buffers and MiniMIPS control networks must be created. We use the elasticized MiniMIPS microprocessor control network as in [8], for modeling and verification with identical considerations for elasticization. As described in Figure 27, elastic MiniMIPS control network consists of 11 elastic buffers, 9 forks, and 6 joins.

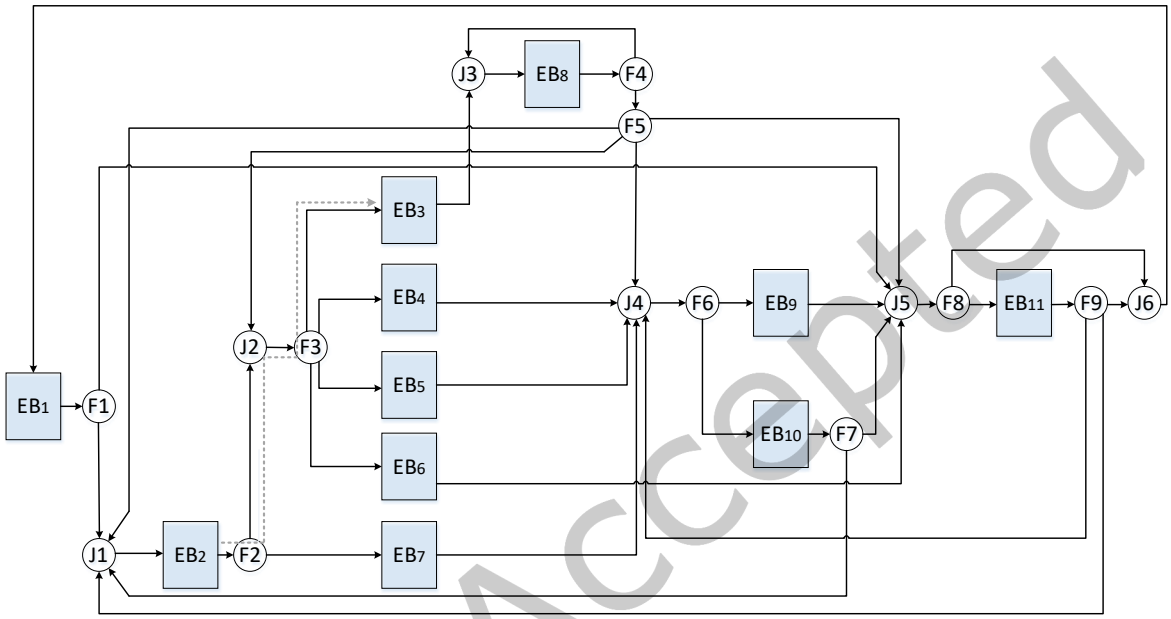


Figure 27: Control network of elastic MiniMIPS [8]

As shown in the previous section, EF and LF_{10} in forks and LJ_{0010} and LJ_{1111} in joins exhibit the best performance in terms of functional verification and timing verification; thereby different combinations of the mentioned components are used to evaluate the MiniMIPS microprocessor. Due to the high-level modeling of this method, various architectures could be investigated in terms of functional and timing verification. Generally, for MiniMIPS using the EB [24] and two types of forks and two types of joins, there are 2^{15} available architectures, five of which are introduced in Table 5.

Table 5: Sample architectures of elastic MiniMIPS

Elastic Component	Architecture 1	Architecture 2	Architecture 3	Architecture 4	Architecture 5
F1	LF_{10}	EF	LF_{10}	EF	EF
F2	LF_{10}	EF	LF_{10}	EF	EF
F3	LF_{10}	EF	LF_{10}	EF	LF_{10}
F4	LF_{10}	EF	LF_{10}	EF	EF
F5	LF_{10}	EF	LF_{10}	EF	EF
F6	LF_{10}	EF	LF_{10}	EF	LF_{10}
F7	LF_{10}	EF	LF_{10}	EF	EF
F8	LF_{10}	EF	LF_{10}	EF	EF
F9	LF_{10}	EF	LF_{10}	EF	EF
J1	LJ_{0010}	LJ_{0010}	LJ_{1111}	LJ_{1111}	LJ_{1111}
J2	LJ_{0010}	LJ_{0010}	LJ_{1111}	LJ_{1111}	LJ_{1111}
J3	LJ_{0010}	LJ_{0010}	LJ_{1111}	LJ_{1111}	LJ_{1111}
J4	LJ_{0010}	LJ_{0010}	LJ_{1111}	LJ_{1111}	LJ_{1111}
J5	LJ_{0010}	LJ_{0010}	LJ_{1111}	LJ_{1111}	LJ_{1111}
J6	LJ_{0010}	LJ_{0010}	LJ_{1111}	LJ_{1111}	LJ_{1111}
EB1~11	EB[24]	EB[24]	EB[24]	EB[24]	EB[24]

All five architectures in Table 5 have been functionally verified (results are available in Table 6). As seen, all architectures have liveness and persistency properties, though architectures 1 and 3 exhibit deadlock. The reason is the combinational cycle that has occurred between LF and LJ. Therefore, to prevent deadlocks, EF must be used in cases where a Fork is linked to a Join (architectures 2, 4 and 5 abide by this architectural rule). Based on the results of fork elements' performance evaluation (Figures 16 - 18), LF components perform better than EF components, thus it can be concluded that architecture 5, due to the usage of LFs, exhibits a better performance (though architectures 2, 4, and 5 are all deadlock free).

Table 6: Functional verification results of elastic MiniMIPS

Elastic MiniMIPS	Persistency	Deadlock freedom	Liveness
Architecture 1	✓	×	✓
Architecture 2	✓	✓	✓
Architecture 3	✓	×	✓
Architecture 4	✓	✓	✓
Architecture 5	✓	✓	✓

To investigate the timing verification of MiniMIPS architecture, we have studied the 5th architecture. For timing verification, paths between two joint EBs must be evaluated. Thirty-one paths are detected on elastic MiniMIPS control networks between EBs, for example, EB2, F2, J2, F3, and EB3 take place in the EB2→EB3 path (dotted line in Figure 27). All available paths are time-verified by variation considerations. Timing verifications for MiniMIPS microprocessors are available in Table 7. Based on the obtained results, the probability of malfunction when the V_{th} variation is set to 20% and 25% is zero, but it increases to 0.1% with variation set to 30%. The malfunctioning probability increases to 0.2% with V_{th}=35%.

Table 7: Timing verification results of elastic MiniMIPS

Path	V_{th} = 20%	V_{th} = 25%	V_{th} = 30%	V_{th} = 35%
$EB_1 \rightarrow EB_2$	1	1	1	0.9990
$EB_1 \rightarrow EB_1$	1	1	1	1
$EB_1 \rightarrow EB_{11}$	1	1	1	1
$EB_2 \rightarrow EB_7$	1	1	1	0.9990
$EB_2 \rightarrow EB_3$	1	1	1	1
$EB_2 \rightarrow EB_4$	1	1	1	1
$EB_2 \rightarrow EB_5$	1	1	1	1
$EB_2 \rightarrow EB_6$	1	1	1	1
$EB_3 \rightarrow EB_8$	1	1	0.9990	0.9990
$EB_4 \rightarrow EB_9$	1	1	0.9990	0.9980
$EB_4 \rightarrow EB_{10}$	1	1	0.9990	0.9980
$EB_5 \rightarrow EB_9$	1	1	0.9990	0.9980
$EB_5 \rightarrow EB_{10}$	1	1	0.9990	0.9980
$EB_6 \rightarrow EB_{11}$	1	1	1	1
$EB_6 \rightarrow EB_1$	1	1	1	1
$EB_7 \rightarrow EB_9$	1	1	0.9990	0.9980
$EB_7 \rightarrow EB_{10}$	1	1	0.9990	0.9980
$EB_8 \rightarrow EB_8$	1	1	1	0.9990
$EB_8 \rightarrow EB_3$	1	1	1	1
$EB_8 \rightarrow EB_4$	1	1	1	1
$EB_8 \rightarrow EB_5$	1	1	1	1
$EB_8 \rightarrow EB_6$	1	1	1	1
$EB_8 \rightarrow EB_9$	1	1	1	1
$EB_8 \rightarrow EB_{10}$	1	1	1	1
$EB_8 \rightarrow EB_{11}$	1	1	1	1
$EB_8 \rightarrow EB_1$	1	1	1	1
$EB_9 \rightarrow EB_{11}$	1	1	1	0.9990
$EB_9 \rightarrow EB_1$	1	1	1	1
$EB_{10} \rightarrow EB_{11}$	1	1	1	1
$EB_{10} \rightarrow EB_1$	1	1	1	1
$EB_{10} \rightarrow EB_2$	1	1	1	1
$EB_{11} \rightarrow EB_9$	1	1	1	1
$EB_{11} \rightarrow EB_{10}$	1	1	1	1
$EB_{11} \rightarrow EB_2$	1	1	1	0.9990
$EB_{11} \rightarrow EB_1$	1	1	1	1

5 CONCLUSION AND FUTURE WORK

Synchronous elastic circuits present a proper solution to solve the existing challenges in synchronous and asynchronous circuits, and their applications are being examined in different areas. Verification is one of the lesser focused subjects in this area. In this paper, a simulation platform and high-level verification were presented using xMAS by considering process variations. Different implementations for elastic components were presented and compared in terms of performance and in the presence of variations. It is shown that internal delays, in most cases, obey Log-normal distributions. By using different properties of distribution functions in the SAN model, each xMAS model is translated to SAN, where the internal delay distribution functions are also included. Finally, each elastic circuit is evaluated in terms of timing and

functionality. The obtained results indicate that even though elastic circuits perform better in the presence of variations (in comparison with synchronous circuits), more investigations are needed in that regard. Performance analysis, and performance distributions could be added to the presented model for future works, so that this platform could be available for both verification and performance analysis.

ACKNOWLEDGMENT

This research was in part supported by a grant from the Institute for Research in Fundamental Sciences (IPM) (Grant No. CS1401-4-77).

REFERENCES

[1] MICHELOGIANNAKIS, G. and DALLY, W.J. 2011. Elastic buffer flow control for on-chip networks. IEEE Transactions on Computers 62, 295-309.

[2] SEITANIDIS, I., PSARRAS, A., KALLIGEROS, E., NICOPOULOS, C. and DIMITRAKOPOULOS, G. 2014. ElastiNoC: A self-testable distributed VC-based network-on-chip architecture. In 2014 Eighth IEEE/ACM International Symposium on Networks-on-Chip (NoCS) IEEE, 135-142.

[3] EDWARDS, S.A., TOWNSEND, R., BARKER, M. and KIM, M.A. 2019. Compositional dataflow circuits. ACM Transactions on Embedded Computing Systems (TECS) 18, 1-27.

[4] YOU, J., XU, Y., HAN, H. and STEVENS, K.S. 2008. Performance evaluation of elastic gals interfaces and network fabric. Electronic Notes in Theoretical Computer Science 200, 17-32.

[5] MAMAGHANI, M.J., KRSTIC, M. and GARSIDE, J. 2016. Automatic clock: A promising approach toward GALSification. In 2016 22nd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC) IEEE, 83-84.

[6] JOSIPOVIĆ, L., GHOSAL, R. and IENNE, P. 2018. Dynamically scheduled high-level synthesis. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 127-136.

[7] DEMME, J. 2021. Elastic Silicon Interconnects. In: Latte'21.

[8] KILADA, E.W.R. 2012. Area, power and performance optimization algorithms for elastic circuit control networks. PhD Thesis, The University of Utah.

[9] LI, C.-H., COLLINS, R., SONALKAR, S. and CARLONI, L.P. 2007. Design, implementation, and validation of a new class of interface circuits for latency-insensitive design. In 2007 5th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE 2007) IEEE, 13-22.

[10] CORTADELLA, J. and KISHINEVSKY, M. 2007. Synchronous elastic circuits with early evaluation and token counterflow. In 2007 44th ACM/IEEE Design Automation Conference IEEE, 416-419.

[11] KILADA, E. and STEVENS, K.S. 2010. Design and verification of lazy and hybrid implementations of the SELF protocol. In IFIP/IEEE International Conference on Very Large Scale Integration-System on a Chip Springer, 206-232.

[12] BAI, Y. 2016. Model-based Design of Embedded Systems by Desynchronization. PhD Thesis, University of Kaiserslautern.

[13] SRINIVASAN, S.K., SARKER, K. and KATTI, R.S. Token-aware completion functions for elastic processor verification. Journal of Electrical and Computer Engineering 2009.

[14] MIRZAEI, M., MOSAFFA, M. and MOHAMMADI, S. 2015. Variation-aware approaches with power improvement in digital circuits. Integration 48, 83-100.

[15] BLAAUW, D., CHOPRA, K., SRIVASTAVA, A. and SCHEFFER, L. 2008. Statistical timing analysis: From basic principles to state of the art. IEEE Transactions on computer-aided design of integrated circuits and systems 27, 589-607.

[16] ADL, S.M.T., MIRZAEI, M. and MOHAMMADI, S. 2018. Elastic buffer evaluation for link pipelining under process variation. IET Circuits, Devices & Systems 12, 645-654.

[17] CHATTERJEE, S., KISHINEVSKY, M. and OGRAS, U.Y. 2012. xMAS: Quick formal modeling of communication fabrics to enable verification. IEEE Design & Test of Computers 29, 80-88.

[18] VAN WESEL, P. and SCHMALTZ, J. 2018. Formal micro-architectural analysis of on-chip ring networks. In 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC) IEEE, 1-6.

[19] LU, Z. and ZHAO, X. 2017. xMAS-based QoS analysis methodology. IEEE Transactions on computer-aided design of integrated circuits and systems 37, 364-377.

[20] BURNS, F., SOKOLOV, D. and YAKOVLEV, A. 2016. A structured visual approach to GALS modeling and verification of communication circuits. IEEE Transactions on computer-aided design of integrated circuits and systems 36, 938-951.

[21] VERBEEK, F., YAGHINI, P.M., EGHBAL, A. and BAGHERZADEH, N. 2016. Deadlock verification of cache coherence protocols and communication fabrics. IEEE Transactions on Computers 66, 272-284.

[22] CORTADELLA, J., KISHINEVSKY, M. and GRUNDMANN, B. 2006. SELF: Specification and design of synchronous elastic circuits. In International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU).

[23] MICHELOGIANNAKIS, G. 2012. Energy-efficient flow control for on-chip networks. PhD Thesis, Stanford University.

[24] CARMONA, J., CORTADELLA, J., KISHINEVSKY, M. and TAUBIN, A. 2009. Elastic circuits. IEEE Transactions on computer-aided design of integrated circuits and systems 28, 1437-1455.

[25] MICHELOGIANNAKIS, G. and DALLY, W.J. 2013. Elastic Buffer Flow Control for On-Chip Networks. IEEE Transactions on Computers 2, 295-309.

[26] CARLONI, L.P., MCMILLAN, K.L. and SANGIOVANNI-VINCENTELLI, A.L. 2001. Theory of latency-insensitive design. IEEE Transactions on computer-aided design of integrated circuits and systems 20, 1059-1076.

[27] GEBHARDT, D. and STEVENS, K.S. 2008. Elastic flow in an application specific network-on-chip. Electronic Notes in Theoretical Computer Science 200, 3-15.

[28] CHATTERJEE, S., KISHINEVSKY, M. and OGRAS, U.Y. 2010. Quick formal modeling of communication fabrics to enable verification. In 2010 IEEE International High Level Design Validation and Test Workshop (HLDVT) IEEE, 42-49.

[29] SANDERS, W.H. and MEYER, J.F. 2000. Stochastic activity networks: formal definitions and concepts⋆. In School organized by the European Educational Forum Springer, 315-343.

[30] DEAVOURS, D.D., CLARK, G., COURTNEY, T., DALY, D., DERISAVI, S., DOYLE, J.M., SANDERS, W.H. and WEBSTER, P.G. 2002. The Mobius framework and its implementation. IEEE Transactions on Software Engineering 28, 956-969.

[31] SOKOLOV, D., KHOMENKO, V. and MOKHOV, A. 2016. Workcraft: Ten years later. This asynchronous world. Essays dedicated to Alex Yakovlev on the occasion of his 60th birthday, 269-293.

[32] Workcraft Website. Available: https://workcraft.org/.

[33] KAUPPILA, A.V. 2012. Analysis of parameter variation impact on the single event response in sub-100nm CMOS storage cells.

[34] ALIOTO, M., PALUMBO, G. and PENNISI, M. 2009. Understanding the effect of process variations on the delay of static and domino logic. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 18, 697-710.

[35] MOSAFFA, M., MOHAMMADI, S. and SAFARI, S. 2016. Statistical analysis of asynchronous pipelines in presence of process variation using formal models. Integration 55, 98-117.

[36] MIRZAEI, M. and MOHAMMADI, S. 2021. Low-power and variation-aware approximate arithmetic units for Image Processing Applications. AEU-International Journal of Electronics and Communications 138, 153825.

[37] HAZEWINKEL, M. 2001. Maximum-likelihood method. Encyclopedia of Mathematics, ISBN 1-4020-0198-3

[38] WESTE, N.H. and HARRIS, D. 2015. CMOS VLSI design: a circuits and systems perspective. Pearson Education India.

[39] PATTERSON, D.A. and HENNESSY, J.L. 2004. Computer organization and design: The hardware/software. In Computer Organization and Design, The Hardware/Software Interface Morgan Kaufmann.

[40] HARRIS, D. and HARRIS, S.L. 2010. Digital design and computer architecture. Morgan Kaufmann.