Metodología de diseño lógico redundante para escenarios con ruido extremadamente alto y bajo voltaje de alimentación

Tesis presentada para obtener el título de Doctor por:
Lancelot García Leyva

Directores:
Dr. Antonio Rubio
Dr. Antonio Calomarde

High Performance IC Design Group
Dpt. Electronic Engineering
Universitat Politècnica de Catalunya
Barcelona Tech

Barcelona, Noviembre 2015 ©

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1

Resumen/Abstract

Resumen

En escenarios futuros de baja potencia y bajo voltaje los sistemas electrónicos presentaran una alta relación de errores suaves o transitorios debido a una drástica relación señal a ruido. Estos errores transitorios pueden afectar los resultados lógicos en un forma permanente. En esta tesis, se ha mostrado una nueva lógica basada en múltiples líneas redundantes para cada nodo lógico como una alternativa para las estrategias basadas en triple redundancia (TMR) dentro de un escenario tolerante a fallas.

La distribución de probabilidades de voltajes en un nodo digital ruidoso, puede ser descrito como la unión de dos distribuciones Gaussianas centradas alrededor del 0-lógico y 1-lógico, que en términos de tensión son 0 y V_{DD}, respectivamente. Un circuito digital presenta un error cuando un valor lógico es mal interpretado, es decir, cuando un 1-lógico es interpretado como un 0-lógico y viceversa, esto esta definido por la probabilidad de error Pe. Una posible forma de reducir la probabilidad de error para un nodo digital ruidoso es por medio del incremento del número de puertos (redundancia de puertos (PR-N)), de tal manera que la probabilidad de error se reduce a medida que se incrementa el número de puertos. En un escenario con un nivel muy alto de ruido, por ejemplo para un SNR de -3dB, la lógica convencional presenta una perdida de fiabilidad aproximada del 50%. Con base en los análisis de fiabilidad previos, en este trabajo de investigación se presenta un nuevo paradigma de diseño nombrado Lógica Tortuga, el cual se basa en la redundancia complementaria de líneas para cada nodo lógico.

En este trabajo de tesis se demostró que una relación adecuada entre redundancia, complejidad y penalización en hardware, es que todos los puertos de los circuitos hagan uso de una redundancia de uno en sus puertos (PR-1). Esta redundancia de puertos PR-1 resulta en que el número de puertos de entrada y salida sea duplicado. Esto es, si se tiene un circuito lógico sin redundancia con dos entradas lógicas a y b; y una salida lógica c, al aplicar una redundancia PR1 se obtiene como resultado un circuito con cuatro puertos lógicos de entrada y dos puertos lógicos de salida. La redundancia de puertos se puede realizar de varias maneras, por ejemplo por medio de la replicación de sus mismos puertos lógicos (aa, bb y cc) o replicando sus puertos lógicos complementarios ($a\bar{a}$, $b\bar{b}$ y $c\bar{c}$). Los cuales pueden ser cualquier combinación de puertos lógicos como puede ser; o cualquier otra combinación.

Tomando como base la replicación de puertos PR1, la metodología de diseño que se usa en esta tesis, consiste en el análisis de la coherencia lógica entre las señales replicadas mediante sus puertos lógicos verdadero y complementario, ya sea una puerta lógica, circuito combinacional, circuito secuencial o procesador. Coherencia lógica es que los valores lógicas de cada entrada y cada salida siempre tengas valores lógicos complementarios, cualquier otro caso significa que ha ocurrido al menos un error, generando tres modos de operación lógica:

- Primer modo de operación: Cuando los puertos de entrada presentan valores complementarios, entonces, el elemento lógico considera que la entrada lógica es correcta, forzando a los valores lógicos complementarios apropiados de salida de acuerdo a la función lógica que ha sido diseñado.
- Segundo modo de operación: Las entradas complementarias tienen el mismo valor lógico, entonces, el elemento lógico identifica estos valores como incoherentes o erróneas, haciendo que los puertos complementarios de salida permanezcan en sus valores previos correctos, evitando que se propaguen valores incorrectos de las entradas a las salidas.
- Tercer modo de operación: En caso de error en ambas señales lógicas de cualquier puerto de entrada replicado (doble error). Entonces el elemento lógico asume que los valores lógicos complementarios de entrada son validas y consecuentemente puede generar un error en sus salidas complementarias redundantes. Sin embargo, la probabilidad de que ocurran eventos simultáneos y complementarios es muy pequeña como se demuestra en el capítulo 2.

Este concepto lógico es llamado Lógica Tortuga debido a que imita el

comportamiento de las tortugas en la naturaleza. Esto es cuando las tortugas se encuentran en una situación de peligro, ellas permanecen en un estado de auto protección. Cuando la amenaza ha desaparecido las tortugas continúan con su vida normal. Siguiendo esta analogía cuando existe ruido que provoque que las señales redundantes complementarias presenten valores lógicos incoherentes, entonces el dispositivo (puerta lógica, circuito combinacional, elemento secuencial o procesador) mantiene sus valores lógicos de salida previos correctos, y cuando el ruido tiene una magnitud tal que no provoque una incoherencia en algunas de las entradas complementarias, entonces los dispositivos tortuga continúan con su operación lógica normal para la cual fueron diseñados.

Cualquier función lógica puede ser implementada siguiendo la metodología de diseño de la lógica Tortuga (principios de coherencia lógica a partir de una redundancia PR1). En orden de construir los elementos lógicos básicos mediante la lógica Tortuga se implementaron las puertas lógicas NOT, NAND2, NOR2 y XOR2 Tortuga, mediante dos posibles formas de construcción a nivel puerta lógica y a nivel transistor, los cuales experimentan diferentes compromisos de diseño y fiabilidad. El diseño a nivel puerta requiere mucho menos tiempo de diseño debido a que los elementos con los cuales se implementa el diseño, son puertas lógicas que se encuentran en la biblioteca de componentes de la estructura de diseño CAD. La implementación a nivel transistor exhibe prestaciones superiores y menor coste en silicio, sin embargo este diseño requiere un dimensionamiento detallado de los transistores.

Con el objetivo de comprobar el mejoramiento de la fiabilidad de las puertas lógicas Tortuga se implementaron puertas lógicas NOT, NAND2, NOR2 y XOR2 mediante diferentes metodologías de diseño enfocadas a la fiabilidad de circuitos lógicos tal como MRF, C-element y Tortuga, tomando como referencia un diseño estándar lógico CMOS. Se realizó una comparativa en materia de coste de hardware y fiabilidad. Donde la lógica Tortuga presenta un mayor coste en hardware mientras que como parámetro de fiabilidad se usa el número de señales espurias presentes en una simulación en el dominio del tiempo. La generación de una señal espuria se realiza mediante la inyección de ruido blanco aditivo Gausiano (AWGN) con una media de 0V y una desviación estándar de 0.5 Vrms para los niveles lógicos 1 y 0. La generación de las señales de entrada con AWGN es mediante una implementación verilog-A y semillas diferentes para cada fuente de ruido, con el objetivo de generar fuentes de ruido no correladas para cada una de las entradas, mientras que las salidas de cada dispositivo bajo prueba tienen una puerta lógica NOT como carga.

En orden a tener las mismas condiciones de simulación para las tres

técnicas se utilizan todas las posibles combinaciones lógicas de entrada en una simulación transitoria, y se inyectaron señales espurias (señales combinatorias más ruido AWGN) a sus entradas de manera arbitraria en el dominio del tiempo, a partir de ello se contó el número de señales espurias presentes a la salida de cada técnica. El experimento se basa en contar toda señal no esperada que tiene un voltaje inesperado en un intervalo de tiempo $(t - tau)$, donde tau es el tiempo mínimo necesario para que una señal de entrada se transfiera a su salida. Una señal espuria es contada cuando ocurre un voltaje no deseado con una amplitud mayor que $(V_{DD}/2)$ y una duración de tiempo de al menos el 10% del periodo de trabajo.

Con base a los principios que rigen a la lógica Tortuga se diseñaron circuitos combinacionales tal como un sumador completo. Donde se considera el escenario donde una sola puerta lógica tortuga del sumador completo (Full Adder - FA) presenta una incoherencia lógica a sus entradas debido a una señal espuria. Por lo tanto, debido a la naturaleza de las puertas lógicas Tortuga, la puerta lógica que presenta la incoherencia lógica mantiene sus valores previos correctos, evitando la transferencia de información incorrecta. Sin embargo, como las otras puertas lógicas Tortuga del sumador completo se encuentran en condiciones de coherencia entre sus valores de entrada, ellas continúan con su operación lógica normal, causando que las demás puertas reciban y generen valores de salida coherentes, pero con información incoherente y en general el sumador completo este realizando un operación no correcta. Debido a este hecho es necesario un sistema de banderas que indique cuando los datos de salida son correctos o no, evitando la propagación de datos erróneos. Por lo tanto, en cada puerta individual Tortuga es incluida una señal de bandera complementaria la cual determina cuando las entradas de la puerta Tortuga son coherentes o no. El proceso de banderas es independiente de la operación de las puertas Tortuga, las cuales son implementadas mediante una puerta convencional XOR para la parte verdadera y una XNOR para la parte complementaria, que verifican la coherencia de cada entrada redundante. Este mismo principio ocurre entre circuito combinacionales que en conjunto también tiene un sistema de banderas que indican cuando el valor lógico de salida es correcto o no.

Con el objetivo de construir circuitos secuenciales es necesario entender la problemática de fiabilidad en circuitos lógicos. En ese sentido es importante entender que las fluctuaciones en la lógica digital de circuitos integrados pueden afectar a los estados lógicos de sus nodos, resultando en algunas ocasiones en el cambio de estado de sus valores lógicos. De esta manera, se genera un efecto transitorio que puede corromper la lógica dentro de un circuito. Si un evento simple transitorio es capturado por un determinado elemento lógico, entonces se puede generar un error suave. Los latchs y Flip-Flops son vulnerables a eventos simples transitorios (SET) du-

rante el tiempo de latcheo (T_{setup}) de un periodo T (ciclo opaco), durante el cual un fault transitorio puede ocurrir. Estos tiempos son llamados como la ventana de vulnerabilidad (Window of Vulnerability-WOV), el cual es el tiempo donde un fault afecta un nodo dentro del latcheo y se propaga a su salida.

Con el objetivo de emular el comportamiento de futuras tecnologías se realizó la experimentación con fuentes de ruido térmico, ruido flicker y ruido de disparo. Estos ruidos o factores son críticos para aplicaciones en el ámbito de las comunicaciones, navegación y control. Realizando una comparación de la tolerancia al ruido para un Flip-Flop D, por medio de la Distancia de Kullback-Leibler, que determina la discrepancia entre la distribución de probabilidades de la salida ruidosa respecto a la distribución de probabilidades de la salida ideal, se obtuvo que la lógica Tortuga es 2.3X, 4.4X, 5.5X y 13.1X, mejor que las técnicas MRF reinforcer, 1st-order CTMR, TMR y estándar CMOS, respectivamente. En el mismo sentido una implementación de una Flip-Flop D presenta respectivamente 2.3X, 4.6X, 6.12X y 13.1X veces mejor robustez al ruido que las técnicas MRF reinforcer, 1st-order CTMR, TMR y estándar CMOS.

Se ha mostrado que los elementos secuenciales basados en la Lógica Tortuga exhiben mucho mejor inmunidad al ruido de acuerdo a la medida de Kullback-Leibler. Estos circuitos presentan un coste adicional en área respecto a una implementación CMOS estándar, pero mucho menor respecto a las técnicas basadas en estrategias probabilísticas, así como a una arquitectura Von Neumann. Por lo tanto, elementos basados en la Lógica Tortuga mejoran la fiabilidad de los sistemas en escenarios ruidosos a un coste razonable, llegando a ser una atractiva alternativa.

Finalmente, se considera el caso general donde un procesador es construido con base a elementos básicos, combinacionales, secuenciales diseñados y construidos con base en los principios de la Lógica Tortuga. De donde se determina que un Procesador Tortuga hereda las propiedades intrínsecas de la misma Lógica. Los cuales de manera general son detener el procesamiento de datos, reloj y señales de control, ya sea por la incoherencia en sus entradas externas o internas, de cualquier modulo o nivel de procesamiento, no así de manera global y/o total de sus elementos o entre procesadores. De lo anterior se determina que es necesario un protocolo de comunicación específico Handshake que asegure los principios lógicos de comunicación entre procesadores síncronos con los mismos principios de la Lógica Tortuga.

Para el escenario de trabajo de un procesador Tortuga se realizan las pruebas de funcionalidad y fiabilidad operando entre dos procesadores Tortuga de características similares (procesador 1 y procesador 3). Como medio

de prueba para el protocolo de comunicación Handshake se utiliza un procesador Tortuga con cuatro estados pipeline. A este procesador le es adaptado el protocolo de comunicación de Sparso, el cual utiliza un esquema tradicional de control acknowledge-Ack y request-Req para la comunicación entre máquinas síncronas vecinas.

Se substituyen las señales de control Ack y Req por señales lógicas redundantes complementarias de acuerdo a la Lógica Tortuga, las cuales verifican el adecuado control de comunicación y correcto procesamiento-cómputo de datos, en los cuatro etapas pipeline del Procesador Tortuga. Cuando ocurre una discrepancia ya sea en las señales de Handshake, elementos redundantes de la lógica combinacional, banderas redundantes o dispositivo de control, se generan señales de alerta tanto en el estado pipeline previo y siguiente. Generando señales de alerta para los procesadores 1 y 3, indicando que el dato actual dentro de la cadena de procesamiento puede ser erróneo y por lo tanto debe desecharse para el procesador 3 y debe repetirse por el procesador 1. De esta manera el procesador 1 genera el dato previo correcto y es enviado al procesador Tortuga en el periodo de reloj siguiente. En el mismo sentido, el procesador 3 desecha el dato actual proveniente del Procesador Tortuga ya que corresponde a un dato incorrecto. Resultado de la adaptación del protocolo de comunicación Handshake de Sparso en un procesador Tortuga con cuatro estados pipeline se obtienen los módulos de lógica combinacional Tortuga; de control; de Registro de almacenamiento temporal RegH; y banderas que indican si los datos de la lógica combinacional Tortuga son correctos.

Como medio de prueba y evaluación del comportamiento de la Lógica Tortuga, se utiliza un multiplicador Baugh-Wooley de 8x8-bits con signo complemento a dos, en una estructura de diseño pipeline. Se utiliza este tipo de multiplicadores ya que es un multiplicador que se implementa mediante una estructura regular, lo que lo hace apropiado para visualizar el comportamiento de la Lógica Tortuga, por encima de una posible optimización de las prestaciones del multiplicador, cuya optimización no es el objetivo de este trabajo de tesis.

El multiplicador Baugh-Wooley es implementado por medio de cuatro estados pipeline. Cada estado pipeline procesa 4 bits de la operación de multiplicación. Dos metodologías son probadas mediante el multiplicador Baugh-Wooley, la convencional y Lógica Tortuga. Los multiplicadores bajo prueba se implementan en las mismas condiciones ya que permite una comparación directa en términos de fiabilidad:

- Dos procesadores de ayuda, el primero suministra los datos de entrada al multiplicador bajo prueba, y el segundo es el que recibe los datos

procesados por el mismo multiplicador.

- Un protocolo de comunicación Handshake, basado en dos señales de control, Req y Ack.
- Cuatro estados pipeline, en donde cada estado procesa 4 bits de la operación de multiplicación.
- Tecnología CMOS de 90nm.
- En el primer experimento se inyectan espurios a las líneas primarias de entrada de los dos multiplicadores.
- En el segundo experimento se inyecta ruido aleatorio en los nodos internos de la segunda etapa pipeline de ambos multiplicadores. Se inyectan espurios en las líneas del bit menos y más significativo del primer y segundo nivel pipeline de ambos multiplicadores.
- Cada experimento implica 1,000 muestras discrepantes simples para ambos multiplicadores y sólo para el multiplicador Tortuga, 1000 muestras discrepantes dobles. El tiempo inicial de cada discrepancia simple y doble, esta localizado de forma aleatoria dentro del periodo T. Las discrepancias inyectadas en el bit menos significativo (LSB) y en el bit más significativo (MSB) son independientes y en instantes de tiempo distintos. Para el caso del multiplicador Tortuga son considerados dos diferentes casos; en el primer caso la discrepancia es inyectada en sólo una de las líneas (línea verdadera del dato) y en el segundo caso la discrepancia es inyectada en ambas líneas del dato. Cada discrepancia es generada con el mismo principio de aleatoriedad y con independencia probabilística, por lo tanto, las señales discrepantes dobles son no correladas.

Un severo modelo de ruido ha sido usado en los experimentos aplicado a una implementación de un multiplicador 8x8-bits pipelined Baugh-Wooley complemento a dos. Los experimentos revelan una tolerancia perfecta para el caso de discrepancias en líneas sencillas (sin errores a la salida de todos los casos) para ambos nodos primarios e internos con un costo de perdida de relojes, entre un 6% y un 25% para el ruido inyectado en los experimentos. Esto significa que las prestaciones del sistema disminuyen sin embargo mantiene un alto nivel de fiabilidad.

Para el mismo experimento, un multiplicador convencional exhibe una alta relación de error entre un 6% y un 48%. La relación de error para la implementación lógica Tortuga propuesta con doble discrepancia en ambas líneas verdadera y complementaria, es menor que el 0.1% cuando el ruido

afecta nodos de entrada primarios y es menor al 0.9% cuando el ruido afecta a nodos internos. Sólo discrepancias simultáneas para ambas líneas pueden generar un error.

De acuerdo a los trabajos de fiabilidad con base a la lógica Tortuga, propuesta y trabajada en esta tesis, se tienen todos los elementos de diseño lógico que permiten integrar cualquier puerta lógica, circuito combinacional, circuito secuencial y procesador.

Abstract

In future scenarios of low power and low voltage the electronic systems will present a high error ratio or voltage fluctuations due to dramatically signal to noise ratio. These transient errors can affect the logical results in a permanent way. In this thesis it has shown a new logic based on multiple redundant lines for each logical node as an alternative to strategies based on triple redundancy (TMR) within a fault-tolerant stage.

The probability distribution of voltages in a noisy digital node can be described, as the union of two Gaussian distributions centred on the 0- and 1-logical, in terms of voltage are 0 and VDD, respectively. A digital circuit has an error when a logical value is misinterpreted, that is, when a 1-logic is interpreted as a 0-logical and viceversa, this is defined by the error probability Pe. One possible way to reduce the error probability for a noisy digital node is by increasing the number of ports n times that in this thesis is called as port redundancy PR-N). In a scenario with a very high noise level, for example for an SNR of -3 dB, conventional logic has a reliability loss of approximately 50%. Based on the reliability analysis in this work thesis, a new design paradigm named Turtle Logic, which is based on the complementary redundancy lines for each logical node, is presented.

In this thesis it was demonstrated that an appropriate relationship between redundancy, complexity and hardware penalty is that all ports of the circuits make use of a redundant one in their ports (PR-1). This redundancy of PR-1 ports results in that the number of input and output ports is doubled. That is, if you have a non-redundant logic circuit with two logic inputs a and b; and a logic output c, to implement redundancy PR-1 is obtained results in a logic circuit with four input ports and two output logical ports. Port redundancy can be performed in several ways, for example by replication of the same logical ports (aa, bb y cc) or replicating its complementary logical ports ($a\bar{a}$, $b\bar{b}$ y $c\bar{c}$). Which can be any combination of logic gates

such as; or any other combination. In this thesis it was shown that the most appropriate ports replication in terms of reliability is through replication of the complementary logical port.

Based on PR-1 the design methodology used in this thesis is the analysis of the logical coherence between the signals replicated by their true and complementary logical ports, either a logic gate, combinational circuit, circuit sequential or processor. Logical coherence or consistency is the logical values of each input and output always have complementary logical values, otherwise means has at least one error occurred, generating three modes of operation logic:

- First mode: When the input ports have complementary values, then the logic element considers the logic input is correct, forcing the appropriate complementary logic output values according to logic function as designed.
- Second mode: The additional entries have the same logical value, then the logic element identifies these values as inconsistent or erroneous, causing additional output ports remain in their previous values correct, preventing that incorrect values have been transmitted from inputs to outputs.
- Third mode: In case of error in both logic signals from any port of entry replicated (double error). Then the logic element assumes that the complementary input logic values are valid and consequently may fail in its redundant complementary outputs. However, the likelihood of simultaneous and complementary events is very small as shown in chapter two of this thesis.

This logical concept is called Turtle logic because it's mimics the behaviour of the turtles in the wild life. This is when the turtles are in a dangerous situation; they remain in a state of self-protection. When the threat has disappeared turtles continue their normal life. Following this analogy when there is noise that causes incoherence values on complementary redundant signals, then the device (gate, combinational circuit, sequential element or processor) maintains its correct previous output logic values, and when the noise has a magnitude that does not cause an inconsistency in some of the complementary inputs, then the devices turtle continue their normal logic operation for which they were designed.

Any logic function can be implemented following the methodology of Turtle logic design (PR1 and logical coherence). In order to build the basic

logic elements by Turtle logic principles were built NOT, NAND2, NOR2 and XOR2 gates, these were implemented through two possible ways, transistor and gate level, which have different compromises of design and reliability. The gate level design requires much less design time because the elements are in the CAD design library, while the mplementation at transistor level exhibits superior performance and lower cost in silicon, however this design requires careful sizing of the transistors.

In order to verify the improvement of the reliability of the Turtle logic gates implemented NOT, NAND2, NOR2 and XOR2 by different design methodologies focused on the reliability of logic circuits as MRF, C-element and Turtle, versus standard logic or CMOS design. A comparison was made with regard to hardware cost and reliability. Tortuga logic has a higher hardware cost but better reliability when a parameter of the number of spurious signals on a simulation in the time domain is used. Generating a spurious signal is performed by injecting additive white Gaussian noise (AWGN) with a mean of 0 V and a standard deviation of 0.5 Vrms to logic levels 1 and 0. The generation of noisy input signals is using Verilog-A with different seed for each noise source implementation, and thus uncorrelated noise sources for each of the inputs, whereas the outputs of each divece onder test has a NOT logic gate as load.

In order to have the same simulation conditions for the three techniques all possible logic input combinations in a transient simulation they are used, and spurious signals (combinatorial signals with AWGN noise) to its inputs arbitrarily in the time domain were injected, from this the number of spurious signals present at the output of each technique was counted. The experiment was based on counting all unexpected signal having a unexpected fluctuation voltage in a time interval $(t - \tau)$, where τ is the minimum time required for an input signal is transferred to its output. A spurious signal is counted when an unwanted voltage occurs with amplitude greater than $(V_{DD}/2)$ and time duration of at least 10% of the work period.

Based on the principles governing combinational Turtle logic circuits as well as full adder designed. Which it is considered the scenario where a single Turtle logic gate from full adder (FA) has its inputs a logical inconsistency due to a spurious signal. Therefore, due to the nature of the Turtle logic gates, the logic gate having the logical inconsistency maintains correct their previous values, avoiding transfer of incorrect information. However, like other Turtle logic gates from full adder are able to coherence between its input values, and they continue with their normal logical operation, causing other circuit receive and generate output values consistent, but with the first inconsistent information the full adder is performing a non proper operation. Due to this fact it requires a flag system to indicate when the output data

are correct or not, preventing the spread of erroneous data. Therefore, in each individual Turtle gate is including a flag, which determines when entries are consistent or not. The process of flags is independent of the operation of the Turtle gates, which are implemented by a conventional XOR gate to the real part and an XNOR for the complementary part, which verify the consistency of each redundant input. This same principle occurs between combinational circuit also where a whole flag system has indicating when the logical output value is correct or not.

With the aim of building sequential circuits it is necessary to understand the problems of reliability in logic circuits. In this regard it is important to understand that fluctuations in the digital logic integrated circuit can affect the logic states of the nodes, resulting in some cases in the state change their logical values, which is called as Single Event Transient (SET). Thus, a transient effect can corrupt within a logic circuit. If a SET is captured by a particular logical element, then you can generate a soft error called Single Even Effet (SEE). The latches and Flip-Flops are vulnerable to SEE during the time latching (T_{SETUP}) within a period T (opaque cycle), where a transitional fault may occur. These times are called as the window of vulnerability (WOV), which is the time where a fault affecting a node within the latching time and it can be to output.

In order to emulate the behaviour of future technologies the experimentation sources has built with thermal noise, flicker noise and shot noise. These noises or factors are critical for applications in the field of communications, navigation and control. Making a comparison of noise tolerance for Flip-Flop D, through the Kullback-Leibler distance, which determines the discrepancy between the probability distribution of the noisy distribution regarding the ideal probability distribution, Turtle logic got is 2.3X, 4.4X, 5.5X and 13.1X, better than techniques MRF reinforcer, 1st-order CTMR, TMR and CMOS standard, respectively. In the same way an implementation of a Flip-Flop D presents 2.3X, 4.6X, 13.1X and 6.12X times better robustness in noisy scenario than MRF reinforcer, 1st-order CTMR, TMR and standard CMOS techniques.

It has been shown that sequential logic elements based on Turtle logic exhibit better noise immunity according to the Kullback-Leibler measure. These circuits have an additional cost in area compared to a standard CMOS implementation, but lower about the techniques based on probabilistic strategies and Von Neumann architecture. Therefore, based on Turtle logic elements improve system reliability in noisy scenarios at a reasonable cost, becoming an attractive alternative.

We consider the general case where a processor is built based on Tur-

tle logic principles (basic, combinational and sequential elements). Where, the Turtle processor has the intrinsic properties of the same logic. Which generally they are stop processing data, clock and control signals, either by inconsistency in its internal or external inputs in any module or processing level, globally and/or all of its elements or between processors. From the above it is determined that a specific communication protocol handshake its necessary to ensure the Turtle logical principles of synchronous communication between processors.

Working for the stage of a Turtle processor functionality and reliability it is operating between two similar processors (processor 1 and processor 3). As evidence for the communication protocol handshake a Turtle processor with four pipelined stages is used. This processor is adapted him communication protocol Sparso, which uses a traditional control scheme acknowledge-Ack and request-Req for communication between neighbouring synchronous machines.

Signals Req and Ack with complementary redundant logic control signals according to the Turtle logic are replaced, which verify proper and correct communication control data-processing in the four stages pipeline Turtle processor. When a redundant Handshake signals mismatch occurs, either combinational element, flags are generated in both previous and next stage pipeline that indicated when a output value its correct or not. Generating warning signals for processors 1 and 3, indicating that the current data within the processing chain may be erroneous and therefore should be discarded to the processor 3 and must be repeated by the processor 1. In this way the processor 1 generates correct previous data and is retransmitted to the Turtle processor at next clock period. Similarly, the processor 3 discarded the current data from Turtle processor to corresponding to incorrect data.

Result of adaptation of the Sparso communication protocol handshake a processor with four Turtle combinational logic modules are obtained; control; temporary storage registration (Regh); and flags indicating whether the data of the Turtle combinational logic are correct.

As a means of testing and evaluating the performance of Turtle Logic, a Baugh-Wooley multiplier 8x8-bit signed two's complement in a pipeline structure design is used. This type of multiplier is used because it is a multiplier that is implemented by a regular structure, which makes it appropriate to visualize the behaviour of the Turtle logic, whose performance optimization is not objective of this thesis.

The Baugh-Wooley multiplier is implemented through four stages pipeline.

Each pipelined stage processing 4-bit multiplication operation. Baugh-Wooley multiplier, conventional and Turtle Logic were testes over two methods. The multipliers under test are implemented within the same conditions to a direct comparison in terms of reliability:

- Two processors help, the first one sending periodic data to the Turtle multiplier under test, and the second is the one that receives the data processed by the same multiplier.
- A communication protocol handshake based on two control signals Ack and Req.
- Four pipelined stages, where each stage processes 4 bits of the multiplication operation.
- 90nm CMOS Technology.
- In the first experiment to inject spurious on primary input lines of the two multipliers.
- In the second experiment random noise is injected in the internal nodes of the second stage pipeline of both multipliers. Are injected spurious on lines of least and most significant bit as well as first and second pipelined stage of both multipliers.
- Each experiment involves 1,000 single discrepant samples for both multipliers and only for Turtle multiplier 1,000 double discrepant samples are used. The start time of each single and double discrepancy is located randomly within the period T. Discrepancies injected into the least significant bit (LSB) and the most significant bit (MSB) is separate and distinct time instants. In the case of Turtle multiplier are considered two different cases; in the first case the discrepancy is injected into only one of the lines (data line true) and in the second case the discrepancy is injected into both lines of data. Each discrepancy is generated with the same principle of randomness and probabilistic independence; therefore, uncorrelated sources are used.

Severe noise model has been used in the experiments applied to an implementation of a 8x8-bit pipelined multiplier Baugh-Wooley two's complement. The experiments reveal a perfect tolerance for discrepancies in the case of simple lines (error-off of all cases) for both primary and internal nodes with a cost of clock lost, between 6% and 25% for noise injected in the experiments. This means that system performance decrease yet maintains a high level of reliability.

For the same experiment, a conventional multiplier exhibits a high error ratio between 6% and 48%. The error ratio for Turtle logic implementation proposal double discrepancy in both true and complementary lines, is less than 0.1% when the noise affects primary input nodes and is less than 0.9% when the noise affects internal nodes. Only simultaneous differences for both lines may fail.

According to the works of reliability based on the Turtle logic proposal and worked in this thesis, you have all the elements that integrate logic design any logic gate, combinational circuit, sequential circuit event a processor.

Capítulo 2

Estado del Arte

Durante varias décadas, la cantidad de información que los microprocesadores son capaces de procesar, ha aumentado gracias al escalado de los transistores CMOS que han seguido la ley de Moore [1], la cual establece que la densidad en un chip se duplica aproximadamente cada dos años, ver Figura 2.1. Tanto los dispositivos CMOS que han reducido su tamaño hasta escalas nanométricas [2, 3, 4, 5] han seguido aproximadamente las predicciones estadísticas del ITRS [6].

La inmunidad al ruido en tecnologías sub micrométricas es cada vez más difícil de lograr, debido a una reducción agresiva del tamaño característico de los dispositivos, a voltajes de alimentación cada vez más pequeños que a su vez provocan menores márgenes de ruido, y a una constante reducción en la carga necesaria para procesar un bit. Estas características en conjunto generan ruido de *ground bounce* e IR *drops* [7], ruido térmico [8], *crosstalk* capacitivo e inductivo [9, 10], fuga de carga [11], entre otros. La reducción del tamaño característico conduce a un efecto de disminución en el número de dopantes y portadores en el canal de las regiones activas de un dispositivo, cuyo efecto genera fluctuaciones aleatorias en los parámetros de los dispositivos. Del teorema del "*Límite Central*" [13], la magnitud de las variaciones aleatorias es inversamente proporcional al número de variables aleatorias; los dispositivos cada vez son más pequeños, por lo que las variaciones aleatorias representan un mayor impacto en el funcionamiento de los dispositivos.

En [8] *Sano* investigó en un régimen nanométrico las variaciones en la corriente en transistores MOS, de donde por medio de simulaciones Monte Carlo, encontró que la desviación estándar normalizada de la corriente de

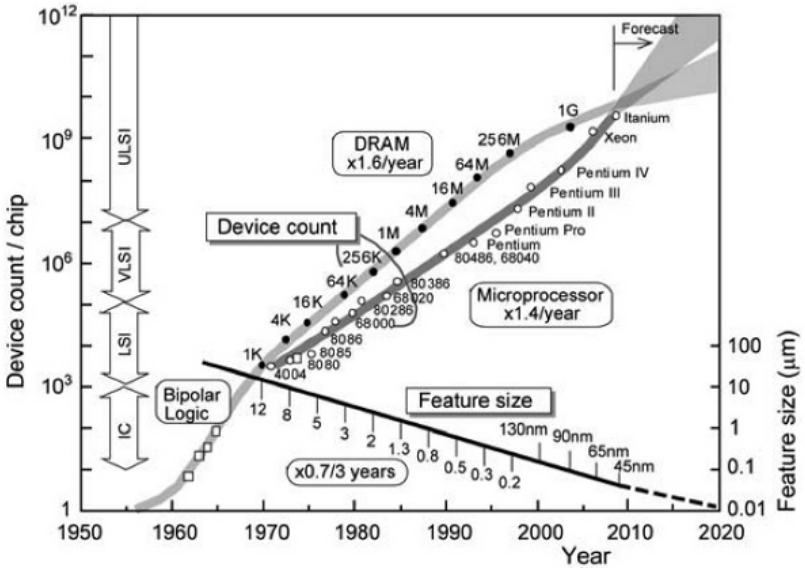


Figura 2.1: Número de transistores (dispositivos) para una CPU en función de las fechas de introducción así como del tamaño característico. La línea corresponde al crecimiento de la cantidad de transistores que ha seguido la ley de Moore [14].

drenador, se incrementa en función de la reducción del ancho de un dispositivo. El comportamiento que presentan las caídas de tensión inductivo y resistivo a través de las redes de distribución de potencia debido al escalado fue estudiado en [7]. *Mezhiba* encontró que la relación de señal-ruido *(Signal to Noise Ratio - SNR)* decrece siguiendo un factor de escalado S para un caso de ruido resistivo y por un factor de S^2 para ruido inductivo. Por lo tanto, un ruido inductivo se incrementa cuadraticamente y llega a ser más significativo con la reducción de la tecnología, que un ruido resistivo. En [21] *Mendoza* mostró que un adecuado compromiso de diseño entre una resistencia y una inductancia, en redes de distribución de potencia en tecnologías nano métricas permite reducir los niveles de ruido en ellas. Así mismo, *Mendoza* mostró que la inmunidad a ruido en circuitos digitales dinámicos disminuirá en tecnologías futuras, principalmente debido al escalado de tensión de alimentación y voltaje de umbral del dispositivo.

En [22] *Borkar* muestra que las variaciones de los parámetros en los procesos de fabricación, así como las variaciones con la temperatura de un dispositivo son más significativas en tecnologías de mayor escalado de integración. Debido a las limitaciones de los procesos de fabricación (longitud de onda del proceso de fabricación, reducidas dimensiones, entre otras), y variaciones en el número de dopantes en dispositivos de canal corto, los

parámetros de un dispositivo, tal como, su longitud (L), su ancho (W), el grueso del oxido (T_{ox}) y el voltaje de umbral (V_{th}) sufren grandes variaciones, que se ven reflejadas en un incremento del retardo y potencia de los circuitos [23].

Teniendo en cuenta el impacto de las variables estadísticas presentadas en el resumen anterior se puede enunciar lo siguiente:

2.1 Problema

Todos las variables que determinan el comportamiento de los dispositivos y circuitos, en materia de tensión, corriente, temperatura están teniendo un comportamiento estocástico, las cuales se encuentran afectadas por las dimensiones atómicas de la materia, por lo tanto, existe la necesidad de nuevas alternativas e innovaciones que hagan frente a escenarios probabilísticos.

Propuestas previas

Se han propuesto algunas técnicas que usan redundancia de hardware para reducir el impacto del comportamiento erróneo de los componentes, y que han logrado incrementar la fiabilidad de los sistemas nanométricos. Las técnicas más relevantes son:

- Lógica tolerante a errores (Error Tolerant Logic) y pueden ser clasificas en:
  - Basadas en la teoría de Von Neumann [24]
    - NAND *multiplexing* [24]
    - N-Tuple Modular Redundancy (NMR) [24]
      - Triple Modular Redundancy (TMR)
      - Cascaded Triple Modular Redundancy (CTMR) [25]
  - Probabilistic CMOS [26]
  - Lógica Probabilística
    - Markov Random Field (MRF) [27]
    - C-Element [28, 12, 15, 29]
    - Stochastic computation [30]

2.2 Técnicas basadas en Von Neumann

En 1950, John Von Neumann presentó un conjunto de técnicas para el diseño de sistemas fiables mediante la utilización de componentes no fiables [24]. Se basa en una estructura de multiplexado considerando dos elementos lógicos, que en su trabajo los llamó votación mayoritaria y puertas NAND. Él duplicó cada puerta lógica NAND N veces remplazando cada una de sus entradas por un paquete de N líneas; de este modo, se obtiene una salida de N líneas, dando como resultado la técnica NAND "Multiplexing", ver Figura 2.2. Para la lógica de la NAND, las entradas del primer paquete son relacionadas con las entradas del segundo paquete de forma aleatoria, como se muestra en la Figura 2.2. El paquete de salidas de las NAND son entregadas al sistema de votación mayoritaria, el cual determina el dato mayoritario que será entregado a la siguiente etapa cómo dato correcto. Von Neumman definió un criterio para decidir el dato mayoritario mediante un nivel Δ definido en el rango $(0 < \Delta < \frac{N}{2})$. Usando una masiva duplicación de los componentes no fiables, Von Neumann concluyó que dicha implementación puede ser fiable, si la probabilidad de fallo de las puertas en conjunto es suficientemente baja. En general, una implementación Von Neumann se realiza por medio de una redundancia N superior a 3 replicas del mismo elemento o dispositivo. Debido a la alta redundancia que presenta está técnica en la práctica los diseñadores de la industria de chips la han utilizado poco. Algunas investigaciones han implementado técnicas que se derivan del trabajo de Von Neumann, dos de las principales son las llamadas "Redundancia Triple Modular" (TMR) y "Redundancia Triple Modular en cascada" (CTMR).

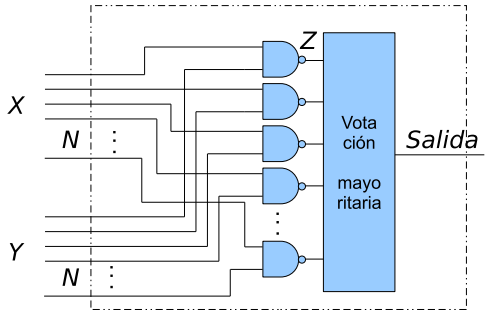


Figura 2.2: Representación esquemática de la técnica NAND multiplexing. X y Y son las señales de entrada y Z es la señal de salida de una puerta lógica NAND, las cuales son replicadas N veces, y cuyas salidas son enviadas a una votación mayoritaria la cual determina cual es el valor de salida correcto.

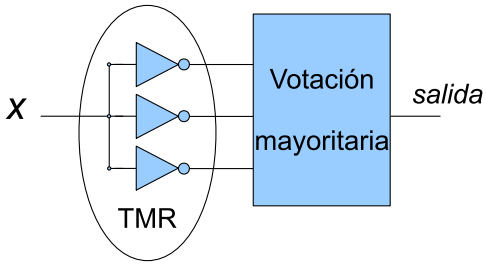


Figura 2.3: Redundancia Triple Modular (TMR) usando como ejemplo el caso de un inversor lógico.

Redundancia Triple Modular

El caso más usado que se deriva de la propuesta de Von Neumann es la técnica de Redundancia Triple Modular (TMR). De donde, la técnica TMR en algunos casos se ha utilizado como referencia para evaluar dispositivos y/o técnicas que soportan una determinada tolerancia a fallos, mientras que en circuitos VLSI se ha utilizado para implementar aplicaciones de alta fiabilidad. El principio de funcionamiento de la técnica TMR, se basa en replicar tres veces cada uno de los módulos, cuya salida es entregada a un módulo de votación mayoritaria que por mayoría determina el dato de salida, ver Figura 2.3.

La Figura 2.3 presenta la técnica TMR en donde se utiliza como lógica de operación un inversor lógico, el cual de acuerdo a los principios que rigen a la lógica TMR este es triplicado, obteniendo tres salidas idealmente iguales desde el punto de vista de fallos en la lógica inversora. Las salidas son entregadas a un votador el cual determina cual es el valor de salida. Cabe hacer mención que esta técnica mejora la fiabilidad de un dispositivo, en este caso. Sin embargo, el votador mayoritario se encuentra bajo las mismas condiciones que la lógica representada, entonces el votador mayoritario tiene las mismas posibilidades de fallar que un módulo simple, y en ese caso el esfuerzo de replicar tres veces los módulos de procesamiento se ve mitigado porque, la salida puede ser correcta o incorrecto de acuerdo a la posibilidad de fallo de salida del votador mayoritario.

Redundancia Triple Modular en Cascada

De acuerdo a los resultados obtenidos por Von Neumann en el trabajo presentado en [24], una forma directa de mejorar la fiabilidad de un dispositivo es replicarlo N veces. De donde la técnica llamada Redundancia Triple Mo-

dular en Cascada (CTMR) se basa en replicar tres veces un circuito TMR, dando como resultado tres bloques TMR o nueve copias del módulo original, por medio de dos niveles en cascada de elementos de votación mayoritaria, los cuales entregan sus salidas a un nuevo votador el cual por mayoría determina el valor de salida. Este proceso puede ser repetido si es necesario, resultando en la técnica de Redundancia Triple Modular en Cascada, llamada en ingles *Cascaded Triple Modular Redundancy* (CTMR), que se muestra en la Figura 2.4. En [25] Spagocci y Fountain muestran que usando CTMR en un nanochip con muchos componentes (10^{11} or 10^{12}) se obtiene una relación de errores extremadamente baja. Una desventaja de CTMR es que introduce un crecimiento exponencial en redundancia respecto a como se incrementan los niveles en cascada aunada a la desventaja heredada de TMR, la cual es que si el último votador mayoritario falla, todo falla.

Las técnicas derivadas del trabajo de Von Neumann mejoran la fiabilidad de los sistema y/o dispositivos, pero todas ellas presentan el problema que si el sistema de votación mayoritaria final falla, todo el esfuerzo que se ha realizado con anterioridad no tendrá relevancia porque todo el sistema fallará.

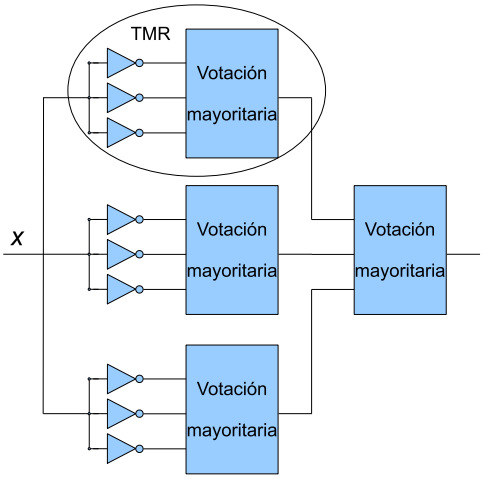


Figura 2.4: Redundancia Triple Modular en Cascada - Cascaded Redundancia Triple Modular (CTMR) usando un inversor como lógica de procesamiento.

2.3 Técnica basada en "C-element"

El "C-element" fue introducido por D. E. Muller en [28] de donde hereda su nombre de "Muller C-element" o simplemente "C-element", el cual tiene

dos entradas (a y b) y una salida (c), de donde su comportamiento lógico se rige como:

- Si, ($a == b$), entonces $c = f(a)$, equivalente a $c = f(b)$.
- Para cualquier otro caso, c permanece sin cambio (se mantiene en su valor lógico previo).

La lógica de un "C-element" asume que una vez que las entradas llegan a tener el mismo valor lógico en sus entradas a y b, estas permanecerán el tiempo suficiente para que la salida c pueda realizar la operación lógica que le permita tomar el valor lógico de su operación. La tabla de verdad que describe el comportamiento lógico del "C-element", se presenta en la Tabla 2.1, de donde, se obtiene un diagrama de estados lógicos, un diagrama de tiempos de operación mismos que se presentan en la Figura 2.5.

Tabla 2.1: Tabla de verdad para un "C-element" con dos entradas (a y b) y una salida (c). c_{n-1} denota una condición de no cambio o que la salida permanece en su estado anterior.

a	b	c
0	0	0
0	1	c_{n-1}
1	0	c_{n-1}
1	1	1

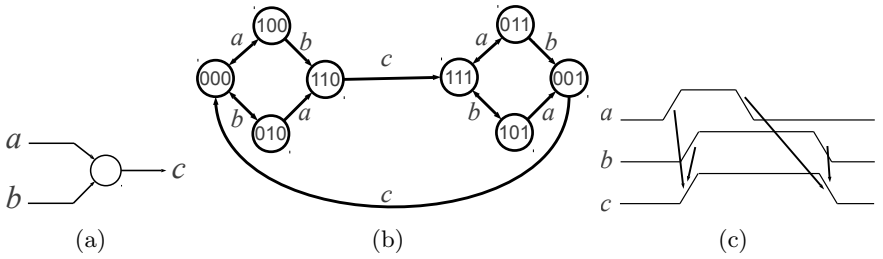


Figura 2.5: "C-Element", (a) Símbolo, (b) Diagrama de estados lógicos y (c) Diagrama de tiempos, de transferencia de la operación comportamental de las entradas a la salida.

El comportamiento lógico de la salida c, de un "C-element" puede ser expresada en términos de las entradas (a y b) y de los estados previos de su salida c_{n-1}, por la función Booleana de la ecuación 2.1.

$$c = ab + (a \oplus b)c_{n-1} \tag{2.1}$$

El "C-element" está estrechamente relacionado con el componente del elemento JOIN de Ebergen presentado en [12, 15] y del RENDEZVOUR de Clarck trabajado en [29]. El componente JOIN tiene ligeramente más restricciones de comportamiento, en el sentido que una entrada no tiene permitido un cambio doble en sucesión. De esta manera, los arcos bidireccionales que se pueden observar en el grafo de estados de la Figura 2.5(a), pueden ser remplazados por un arco unidireccional. En circuitos asíncronos, comúnmente un "C-element" se utiliza como si este fuera un componente JOIN. De hecho, cualquier implementación del "C-element" puede ser usado para realizar un componente JOIN.

De acuerdo a Ebergen en [12, 15] un elemento JOIN esta definido por la ecuación 2.2 y su representación esquemática se presenta en la Figura 2.6. Donde las señales ($a?$ y $b?$) son las señales de entrada y ($c?$) corresponde a la señal de salida, equivalentes en operación a las señales a, b y c del "C-element". La interpretación de la ecuación 2.2 es: la salida $c?$ toma el valor de $a?$, si y solo si, a es igual a b.

$$\mathbf{pref} * [(a?||b?); c!] \tag{2.2}$$

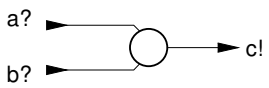


Figura 2.6: Componente básico JOIN de Ebergen presentado en [12, 15].

Una posible clasificación del "C-element" de acuerdo a su implementación es:

- Convencional o "pull-up" "pull-down" [31].
- Dinámico y estático.
- De Retroalimentación débil, y
- Simétrico
- Doble riel

El “C-element” convencional se muestra en la Figura 2.7(a), el cual ha sido presentado en [31] por Sutherland, el cual utiliza un elemento denominado como“keepers”, para almacenar el dato de salida. Los elementos ”keepers” en la Figura 2.7(a) son N3, N4, N5, P3, P4 y P5; los cuales están diseñados con un ancho de canal mínimo W.

El “C-element” dinámico se muestra en la Figura 2.7(b), que constituye la parte funcional básica de una implementación de riel sencillo (single-rail). La versión estática se diferencia de la dinámica sólo en el mecanismo para preservar el estado de salida. Los parámetros de diseño de este circuito son principalmente las dimensiones del canal, y las dimensiones del buffer de salida respecto a las variables U y r, ver Figura 2.7(b).

El “C-element” de retroalimentación débil “weak feedback” es presentado por Martin en [32] y presentado en la Figura 2.7(c). Este circuito utiliza un inversor de retroalimentación débil para mantener el estado de salida el cual amarra el dato hasta que las condiciones de sus entradas deriven en que este cambie.

Una implementación simétrica del “C-element” es presentada por Berkel en [33] y mostrado en la Figura 2.7(d). El “keepers” en esta implementación se implementa mediante los transistores N3 y P3 que son los encargados de mantener el estado de la salida. Esta es una implementación con una relación mínima en los tamaños de los transistores del “keepers”, con la ventaja que es simétrica respecto a las entradas. Por otro lado, Elissati en [34] presenta una comparación en términos de frecuencia contra potencia dinámica de una implementación a nivel transistor, utilizando una tecnología de 65nm para una implementación Convencional, Dinámica, de retroalimentación débil y simétrica de un “C-element”.

Circuitos lógicos diferenciales también llamados circuitos de doble riel usualmente requieren ambas señales de entrada, una señal normal o verdadera y su complemento. Por lo tanto, los circuitos lógicos diferenciales usualmente requieren doble señal, y ellos pueden ser benéficos en términos de velocidad, energía y área. Algunas implementaciones del “C-element” de doble riel se presentan en [35] y no corresponden a las familias que se presentan en [36, 37], debido a que en [36, 37] se usa un inversor latch para mantener los estados de las salidas. Mientras que Shams en [35] hace referencia a esas clasificaciones como una lógica diferencial implementado por medio de un latch inversor.

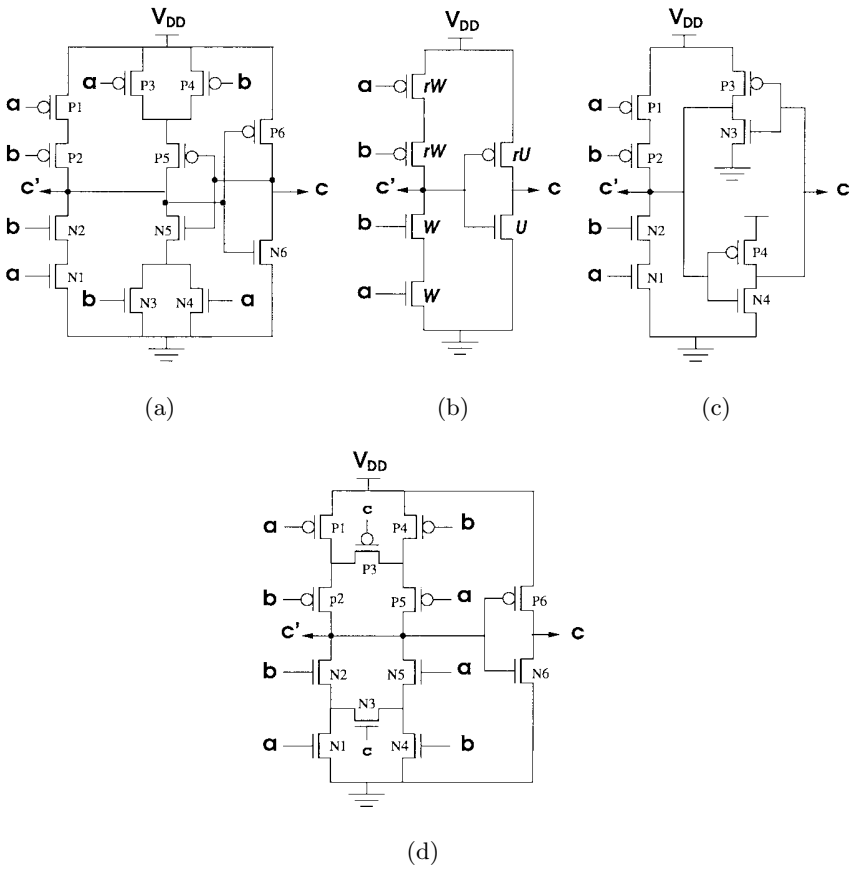


Figura 2.7: Implementación CMOS a nivel transistor de un "C-Element", (a) Convencional, (b) Dinámico, (c) Retroalimentación débil y (d) Simétrica.

2.4 Técnicas probabilísticas

Se puede argumentar que el éxito de toda la industria de semiconductores se ha basado en una abstracción única, fundamental, a saber, que la computación digital se compone de una secuencia determinista de ceros y unos lógicos. Desde el nivel de la lógica Booleana la funcionalidad de un circuito corresponde a la capa física, que produce valores de tensión que se pueden interpretar como una lógica de valores exactos. Esta abstracción ofrece todos los beneficios del paradigma digital: precisión y modularidad. Sin embargo, como los circuitos se están escalando hacia dimensiones nanométricas, los circuitos físicos que sustentan esa abstracción es cada vez más costosa y difícil lograr[38]. En ese nuevo régimen los circuitos lógicos diseñados y fabricados en escalas nanométricas pueden producir errores generados por ruido y cualquier perturbación con las suficiente energía para cambiar el es-

tado lógico de un nodo, especialmente dispositivos electrónicos que operan en condiciones hostiles [39]. Por lo tanto, nuevas visiones probabilistas para la computación digital son apremiantes.

La tarea de analizar circuitos que están operando sobre entradas probabilísticas es tratado por Parker en [40], de donde establece que la correlación de señales de re-convergencia debe ser tomada en cuenta. Parker y Savir en [40] y [41] encontraron que aplicaciones para tareas convencionales, tal como, el análisis de sincronizar señales o de suministrar energía han emigrado a comportamientos probabilísticas como lo trabajaron Liou y Marculescu en [42] y [43].

La síntesis de circuitos para transformar un conjunto de probabilidades dado un conjunto de probabilidades es tratado por Gill en [44] y [45], de donde Gill se enfocó en la síntesis de máquinas secuenciales de estado, motivado por los problemas de computación neuronal.

Jeavons et al. en [46] aplicaron funciones fijas para representar los bits de los estados lógicos actuales en diferentes secuencias, en donde consideraron el problema de transformar secuencias binarias estocásticas a través de lo que ellos llamaron "algoritmos locales". Equivalente a realizar operaciones de bits estocásticos mediante lógica combinacional, por lo que, en esencia es la misma problemática que considera Qian en [30], donde el principal resultado de Qian es un método para generar secuencias con probabilidad $\frac{m}{n^d}$ de un conjunto de secuencias binarias estocásticas, con probabilidades dentro del conjunto $\{\frac{1}{n}, \frac{2}{n}, ..., \frac{n-1}{n}\}$. En contraste al trabajo de Jeovans et al. el objetivo de Qian es minimizar el número de fuentes probabilísticas.

Por otro lado, trabajos partidarios de *CMOS Probabilístico* (PCMOS) presentados en [26, 47], discuten el problema de síntesis de un lógica combinacional para transformar estados lógicos a valores de probabilidad. Los autores sugieren el uso de un circuito basado en un árbol para realizar un conjunto de probabilidades objetivo, sin embargo, en [26, 47] no dan detalles de su implementación.

Wilhelm y Bruck en [48] propusieron una estructura general para sintetizar circuitos lógicos para lograr una probabilidad deseada, circuitos que originalmente fueron discutidos por Shannon en [49]. Estos circuitos consistían en relevadores o dispositivos electromecánicos que funcionan como un interruptor controlado por un circuito eléctrico que podían estar en cualquier estado de los dos estados de abierto o cerrado; los circuitos calculaban un 1 lógico si existía una ruta cerrada a través del circuito, en el cual cada conmutación o cambio lógico tenía una probabilidad de certeza de estar

abierto o cerrado. Wilhelm y Bruck propusieron un algoritmo para generar un circuito de conmutación estocástico para calcular cualquier probabilidad binaria. Zhou y Bruck generalizaron el trabajo de Wilhelm y Bruck en [50], donde consideraron el problema de sintetizar un circuito de conmutación estocástica para realizar una probabilidad arbitraria fraccionaria con base-n $\frac{m}{n^d}$ de un conjunto de conmutaciones probabilísticas $\{\frac{1}{n}, \frac{2}{n}, ..., \frac{n-1}{n}\}$. Así mismo mostraron que cuando n es un múltiplo de 2 o 3, esta realización es posible. Sin embargo, para cualquier número primo n más grande que 3, existe una probabilidad fraccionaria base-n que no puede ser realizada por algún circuito de conmutación estocástico.

En contraste al trabajo de Gill, de Wilhelm y Bruck y el de Zhou y Bruck, Qian considera circuitos combinacionales, los cuales son circuitos sin memoria implementados por puertas lógicas, donde el enfoque de Qian se amolda o apega a una implementación a nivel circuito de la técnica denominada PCMOS [26, 47].

Si se considera el ejemplo que se muestra en la Figura 2.8, donde se proporciona una cadena de bits estocásticos e independientes como entradas, para ser computados por una puerta lógica AND se genera o produce una cadena de bits a la salida, con una probabilidad que es el producto de las probabilidades de la cadena de bits a la entrada.

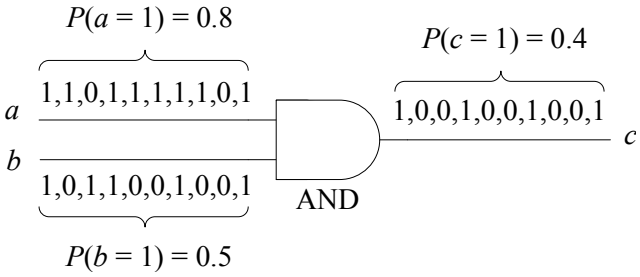


Figura 2.8: Cálculo de probabilidades para una multiplicación de un bit por medio de computación estocástica.

La mayoría de circuitos digitales integrados son diseñados para procesar entradas deterministas de 0 y 1 lógico. Un paradigma alternativo es diseñar circuitos que operen sobre una cadena de bits estocásticos, donde cada cadena representa un número real x ($0 \leq x \leq 1$), que a través de una secuencia de bits aleatorios obtiene una probabilidad x. Entiendace que cuando ocurre un 1 lógico, se tiene una probabilidad de $1-x$ para el caso de un 0 lógico. Por lo tanto, tales circuitos pueden ser vistos como una construcción que acepta probabilidades reales como entradas y obtiene valores de probabilidad reales, como salida.

Qian en [30], propuso un método para sintetizar funciones arbitrarias a través de cálculo sobre cadenas de bits estocásticos, dónde, dichas cadenas de bits estocásticas se generan por medio de una construcción pseudo-aleatoria, por ejemplo, un registro de desplazamiento con retroalimentación. En otras palabras, si existe una fuente que genera valores lógicos aleatorios estos valores pueden ser utilizados de forma directa. Por ejemplo, en [26], los autores proponen una construcción probabilística CMOS nombrada"PCMOS", que genere bits aleatorios de una fuente intrínseca de ruido. En [26, 47], utilizan switches que aplican para generar un Probabilistic system-on-a-chip también conocido como PSOC; este sistema proporciona probabilidades intrínsecas que pueden ser utilizadas en algoritmos probabilísticos.

Para esquemas de cadenas de bits estocásticos que son generadas por fuentes físicas, una importante limitación es el costo de generación de los diferentes valores probabilísticos. Por lo tanto, sí cada valor probabilístico es determinado para un nivel de voltaje especifico, se necesitan diferentes niveles de voltaje para generar diferentes valores probabilísticos. Por lo tanto, para una aplicación que requiera valores diferentes, serán requeridos muchos reguladores de voltaje; lo que lo puede hacer prohibitivo en términos del costo de área y energía.

2.5 Técnica basada en los campos aleatorios de Markov

Los campos aleatorios de Markov se usan en áreas tal como, visión por computadora, física, biología computacional, comunicaciones, entre otros, y en [27] fue propuesto por Bahar como un modelo de incertidumbre y ruido de cómputo. Los campos aleatorios de Markov definen un conjunto de variables aleatorias llamadas *sitios*, definidos por $X = \{x_1, x_2, ..., x_k\}$, donde cada variable x_i puede tomar un conjunto de valores llamados *etiquetas*. En la figura 2.9 se muestra una representación gráfica de los *sitios* en X representados por circunferencias, que están relacionados unos con otros a través de un sistema de vecindario denominado *clique* definido por el conjunto de variables $X - \{x_i\}$. La probabilidad de un *sitio* en particular está determinado por su vecindario, el cual depende sólo de sus vecinos inmediatos los cuales están conectados por un *borde*. El *borde* de un vecindario representa la dependencia condicional entre variables conectadas en un mismo vecindario. Si esto es trasladado a un escenario de en términos de circuitos lógicos, estos nodos representan todas las entradas y salidas de un circuito lógico, como se muestra en el grafo de la Figura 2.10, respecto al circuito combinacional que se ilustra en la misma figura.

Un grafo es descrito mediante las propiedades de los campos aleatorios de Markov, mediante sus nodos que son variables aleatorias lógicas que pueden tomar cualquier valor entre 0 y V_{DD} en el escenario de circuitos lógicos; y mediante su *bordes*, que son las dependencias condicionales entre variables. Es importante hacer notar que no hay una dependencia de causalidad en el flujo de la lógica, sólo dependencia estadística; y que la definición de estado se aplica a los valores que toman las entradas y salidas de un circuito combinacional y no debe confundirse, con la definición convencional de estado de un circuito combinacional. Por ejemplo, el grafo de la Figura 2.10, representa una puerta lógica OR y NOT por medio de cliques, entonces, si la salida de la puerta lógica OR es un 1 lógico, entonces ambas entradas son un 1 lógico. Hay una dependencia estadística entre el estado de la salida y los estados de la entrada. Esta dependencia entre las entradas y las salidas, es modelada con un *borde*.

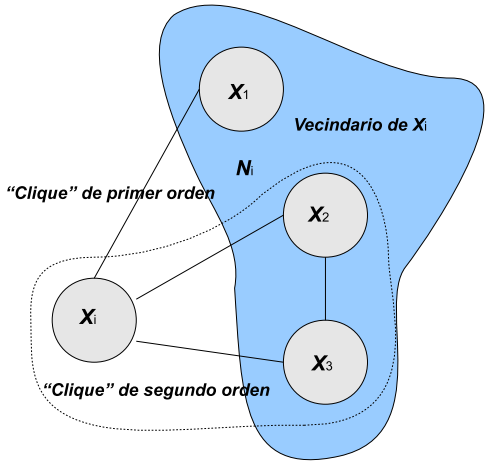


Figura 2.9: Representación de un vecindario MRF por medio de cliques. Cada clique representa una puerta lógica, por ejemplo, el clique de primer orden es un inversor entre X_i y X_1, mientras que un clique de segundo orden es una puerta lógica OR, regida por las variables X_i, X_2 y X_3.

Un ejemplo de la relación entre un circuito digital combinacional y los campos aleatorios de Markov, se presenta en la Figura 2.10. De donde, se toma la premisa de que todas las variables lógicas $\{X_0, X_1, X_2, X_3, X_4, X_5\}$ varían de manera aleatoria entre el rango de los niveles lógicos ideales entre 0 y V_{DD}). La teoría de MRF establece el hecho que los estados lógicos correctos son aquellos que maximizan su probabilidad conjunta de un circuito lógico. A partir del grafo de la figura 2.10 se pueden establecer tres conjuntos de *cliques* definidos como $\{X_0, X_1, X_3\}$ $\{X_2, X_3, X_4\}$ y $\{X_4, X_5\}$, que representan a las tres puertas lógicas XOR, NAND y NOT, respectivamente. Estos tres cliques se presentan en la Figura 2.10 y cuyos valores de estado están determinados por sus funciones booleanas o tabla de verdad,

de acuerdo a la lógica boolena convencional.

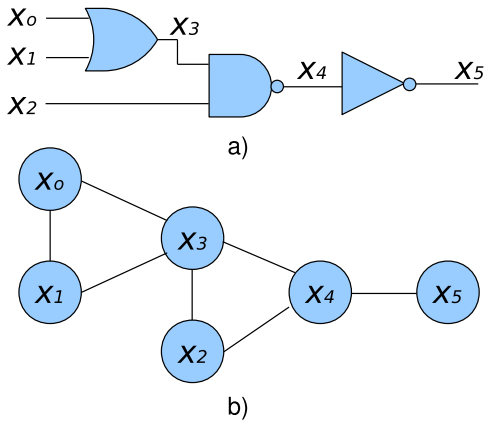


Figura 2.10: Representación de los campos aleatorios de Markov, a) Circuito lógico combinacional mediante tres puertas lógicas y b) Grafo de dependencia del circuito lógico combinacional.

La notación que usa MRF para representar los valores que pueden tomar los nodos de un circuito lógico combinacional, por medio de sus probabilidades se basa en el trabajo de Stuart presentado en [51], donde un conjunto de probabilidades para un nodo es representado como:

$$p(x_1, x_2, ..., x_n) = p(X_1 = x_1, X_2 = x_2, ..., X_n = x_n), \tag{2.3}$$

dónde, $x_i \in X - \{x_i\}$.

Si el conjunto de la variable aleatoria X es estadísticamente independiente entonces la probabilidad conjunta puede ser expresada como el producto de un conjunto de probabilidades definido como:

$$p(x_1, x_2, ..., x_n) = p(x_1)p(x_2)...p(x_n) \tag{2.4}$$

Por lo tanto, para una variable determinada y con fines de ejemplo la probabilidad de la variable x_2 es:

$$p(x_2 \mid x_1, x_2, ..., x_n) = \frac{p(x_1, x_2, ..., x_n)}{p(x_1, x_3, ..., x_n)}, \tag{2.5}$$

Para el caso donde todas las variables aleatorias son independientes,

entonces:

$$p(x_2 \mid x_1, x_2, ..., x_n) = p(x_2). \tag{2.6}$$

2.5.1 Puerta lógica NOT MRF

Con base a los principios lógicos que rigen a la lógica basada en los campos aleatorios de Markov se trata el diseño de una puerta lógica NOT en esta sección.

El circuito que implementa la función lógica NOT por medio del método MRF, es presentada en la Figura 2.12. Considerando una puerta lógica NOT y el grafo de la figura 2.11, x_0 y x_1 corresponden a los nodos de entrada - salida de la misma puerta lógica, que de acuerdo a la lógica combinacional existen cuatro posibles estados lógicos, tal como se presenta en la Tabla 2.2. Cabe resaltar que existen dos estados validos los cuales se referencian mediante $f = 1$, y los estados inválidos mediante $f = 0$. Con lo cual el objetivo de la metodología MRF, es encontrar un circuito que refuerce los estados validos, disminuyendo la probabilidad de los estados inválidos y así el sistema mejore su robustes frente a fallos o errores transitorios. Para lo cual MRF implementa esta acción por medio de una función de retroalimentación, que toma en cuenta los estados validos bajo la premisa de una probabilidad de ocurrencia igual para los estados validos, como es indicado en la Tabla 2.2. Mientras que por otro lado se ignoran los estados inválidos, a los cuales MRF les asigna una probabilidad de ocurrencia cero.

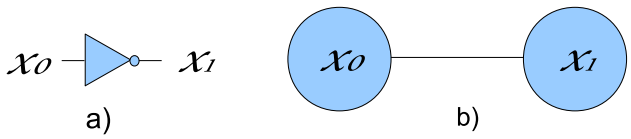


Figura 2.11: Puerta lógica NOT, a) Circuito inversor lógico, b) Grafo de dependencia para la metodología MRF.

Esta implementación en particular incrementa la fiabilidad de puertas lógicas, en un escenario donde todos los nodos del sistema tienen la misma probabilidad de ocurrencia, de acuerdo a los resultados presentados por Bahar en [27].

De un análisis de todos los estados posibles que puede tomar un inversor lógico, implementado mediante la metodología MRF, la puerta lógica NOT MRF presenta errores en su funcionamiento. Para ejemplificar esto,

Tabla 2.2: Todos los posibles estados para una puerta lógica NOT.

x_0	x_1	f	Decimal	Estado	Mint.	Retroalimentación	Prob.
0	0	0	0	I	$m_0 = \overline{x_0} \cdot \overline{x_1}$		0
0	1	1	1	V	$m_1 = \overline{x_0} \cdot x_1$	$x_0 = \overline{m_1}$; $\overline{x_1} = \overline{m_1}$	0.5
1	0	1	2	V	$m_2 = x_0 \cdot \overline{x_1}$	$\overline{x_0} = \overline{m_2}$; $x_1 = \overline{m_2}$	0.5
1	1	0	3	I	$m_3 = x_0 \cdot x_1$		0
I:Estado con errores; V=Estado sin errores							

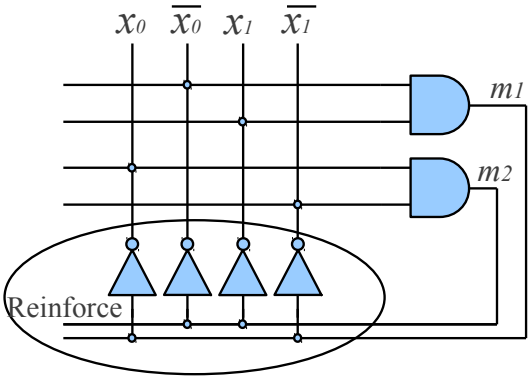


Figura 2.12: Codificación de una función clique de un circuito MRF para dos variables lógicas que definen un Inversor lógico.

se toma el estado de la puerta lógica NOT con valores de estado lógico de $\{x_0\overline{x_0}x_1\overline{x_1}\} = \{0000\}$, entonces, la retroalimentación (ver Figura 2.12) para este caso en particular, en lugar de hacer que los estados incorrectos converjan a un estado correcto, hace que los nodos de la puerta lógica NOT MRF converjan a valores incorrectos de $\{x_0\overline{x_0}x_1\overline{x_1}\} = \{1111\}$. Este estado lógico incorrecto a su vez converge al estado original incorrecto $\{0000\}$, provocando que los valores lógicos de la puerta lógica NOT MRF, entren en un estado lógico nodal oscilatorio, hasta que por alguna razón externa cambien sus estados lógicos a un estado valido. Esto hace que el sistema MRF presente, una no fiabilidad en ambientes donde la puerta lógica NOT MRF se encuentra bajo grandes cantidades de estrés y todos los estados tengan una probabilidad de ocurrencia diferente a cero.

2.5.2 Puerta lógica NAND MRF

Con base a la metodología de diseño lógico MRF hasta ahora presentado, se diseña una puerta lógica NAND MRF, la cual toma como base los puertos lógicos de una puerta lógica NAND clásica, Figura 2.13. Para este

caso existen ocho posibles estados lógicos, que se muestran en la Tabla 2.3. Cada estado lógico es considerado, sin embargo no todos los estados lógicos son validos. Los estados validos en la Tabla 2.3 se identifican por V y los inválidos por I. La retroalimentación que permite reforzar los estados lógicos de acuerdo a los principios lógicos MRF o en su caso que converjan a un estado correcto, es obtenida por la suma de los estados lógicos validos para cada puerta lógica, ver Tabla 2.3 columna retroalimentación. El circuito que resulta de aplicar los principios lógicos MRF se obtien el circuito lógico que hace la misma funcionalidad lógica de una puerta lógica NAND convencional, que presenta en la Figura 2.14.

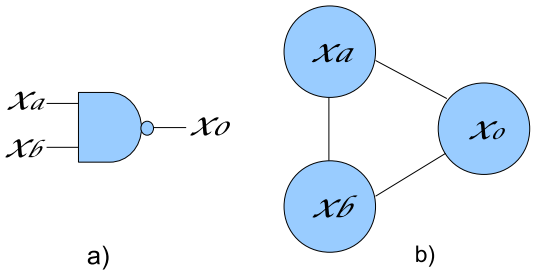


Figura 2.13: Puerta lógica NAND, (a) Circuito lógico NAND, (b) Grafo de dependencia para una puerta lógica NAND mediante la metodología de diseño lógico MRF.

Tabla 2.3: Todos los posibles estados para una puerta lógica NAND MRF.

x_a	x_b	x_o	f	Dec.	Est.	Minterminos	Retroalimentación	Prob.
0	0	0	0	0	I	$m_0 = \overline{x_a} \cdot \overline{x_b} \cdot \overline{x_o}$		0.0
0	0	1	1	1	V	$m_1 = \overline{x_a} \cdot \overline{x_b} \cdot x_o$	$x_a = \overline{m_1 \cdot m_3}$	0.25
0	1	0	0	2	I	$m_2 = \overline{x_a} \cdot x_b \cdot \overline{x_o}$	$\overline{x_a} = \overline{m_5 \cdot m_6}$	0.0
0	1	1	1	3	V	$m_3 = \overline{x_a} \cdot x_b \cdot x_o$	$x_b = \overline{m_1 \cdot m_5}$	0.25
1	0	0	0	4	I	$m_4 = x_a \cdot \overline{x_b} \cdot \overline{x_o}$	$\overline{x_b} = \overline{m_3 \cdot m_6}$	0.0
1	0	1	1	5	V	$m_5 = x_a \cdot \overline{x_b} \cdot x_c$	$x_o = \overline{m_6}$	0.25
1	1	0	1	6	V	$m_6 = x_a \cdot x_b \cdot \overline{x_o}$	$\overline{x_o} = \overline{m_1 \cdot m_3 \cdot m_5}$	0.25
1	1	1	0	7	I	$m_7 = x_a \cdot x_b \cdot x_o$		0.0
I:Estado con errores; V=Estado sin errores								

Esta técnica mejora la fiabilidad de puertas lógicas sin embargo para escenarios de alto ruido presenta la problemática de oscilación debido a que no son tomados en cuenta todos los estados posibles para poder diseñar un sistema. Por lo tanto, si el sistema entra en un estado que no ha sido tomado en cuenta, el sistema fallará. Otra problemática que presenta, es que debido al mismo hecho de no tomar en cuenta todos los estados posibles para el sistema, este converge a un determinado estado no correcto, el cual converge a otro estado incorrecto y este a su vez converge al estado anterior

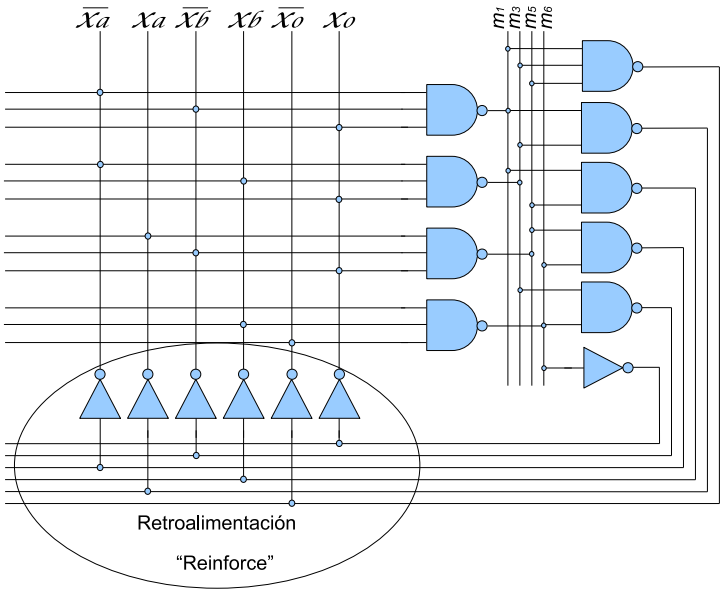


Figura 2.14: Puerta lógica NAND MRF sintetizada a partir de la Tabla 2.3.

no correcto, entrando a un ciclo de oscilación nodal.

Capítulo 3

Origen del problema y escenario de trabajo

Durante varias décadas, la cantidad de datos que los ordenadores son capaces de procesar se ha incrementado entre otros factores debido al producto del escalado de los transistores MOS como puede observarse en la Figura 3.1, esto de acuerdo a las predicciones de la ley de *Moore*, continuamente actualizada y verificada por *The International Technology Roadmap for Semiconductors* (ITRS) [6]. Los dispositivos escalados en un régimen nanométrico [6] así como tecnologías emergentes [3, 4, 5], son periódicamente reportados por el ITRS [6], cuyos dispositivos presentan un comportamiento estadístico. Tales comportamientos, son debido tanto al ruido como las variaciones inherentes en los procesos de fabricación y sus condiciones de operación.

En tecnologías de escala subnanométrica (deep submicron DSM [10, 52]) la inmunidad a ruido llega a ser más difícil de lograr debido a la reducción de los tamaños característicos de los transistores, voltajes de alimentación más pequeños, en otras palabras márgenes de ruido más pequeños y densidad de integración más grande [6], lo que hace que las tecnologías de escala subnanométrica sean inherentemente ruidosas, en materia de: ruido térmico [8], ruido de disparo, ruido Flicker, "ground bounce" e "IR drops" [7], "crosstalk" capacitivo e inductivo [10, 53], fuga de carga y carga compartida [11]. Dado que los dispositivos se están fabricando cada vez más pequeños (Figura 3.1), los ruidos inherentes en un dispositivo debido a las variaciones de fabricación llegan a ser más relevantes. Los principales tipos de ruido en semiconductores son ruido térmico, ruido de disparo y ruido flicker.

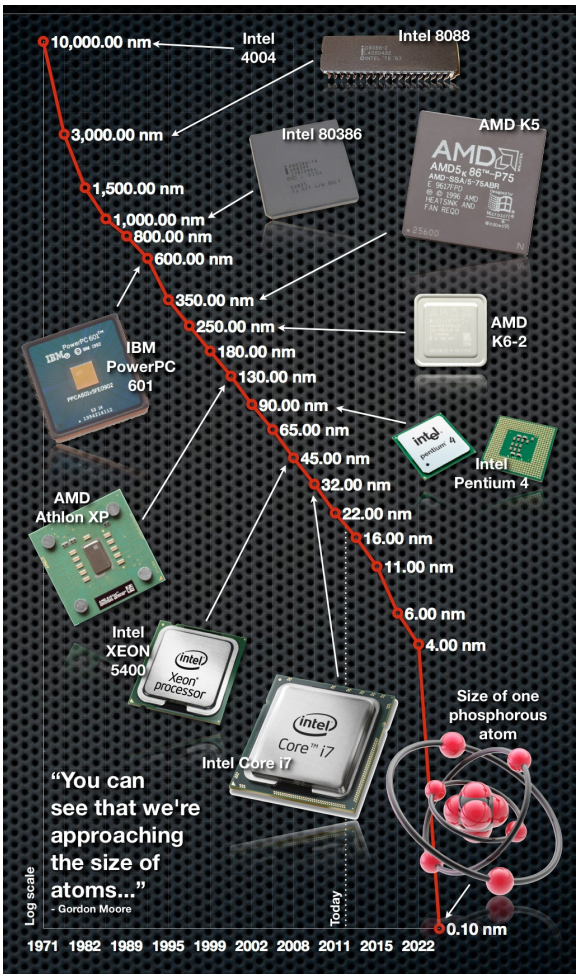


Figura 3.1: Tendencia de las dimensiones de la Tecnología.

El ruido térmico o ruido de Johnson–Nyquist se genera por la agitación térmica de los portadores de carga en equilibrio que sucede con independencia del voltaje aplicado [54]. El ruido térmico es aproximadamente blanco, lo que significa que su densidad espectral de potencia es casi plana, que de acuerdo al teorema del límite central se puede determinar que la amplitud de la distribución del ruido térmico es Gausiana [55]. Por lo tanto, el ruido térmico puede modelarse como una fuente de tensión que representa el ruido de una resistencia no ideal en serie con una resistencia libre de ruido, los cuales tienen una densidad espectral de potencia dada por la ecuación 3.1.

$$\overline{vt^2} = 4kTR\delta f \tag{3.1}$$

donde k es la constante de Boltzmann, T es la temperatura absoluta, R es la resistencia y δf es el ancho de banda en Hz sobre la cual es medido el ruido. Esta ecuación es valida mientras $f << kT/h$, dónde h es la constante de Planck. En cualquier otro caso el voltaje rms del ruido térmico esta dado por:

$$\overline{vt^2} = \frac{4hfR\delta f}{exp(\frac{hf}{kT}) - 1} \tag{3.2}$$

Función de densidad de probabilidad para ruido térmico

En la Figura 3.2 parte inferior se muestra la evolución temporal de un ruido térmico, el cual presenta un comportamiento impredecible en el tiempo. Si a partir de esta señal se genera un histograma, se obtiene una función de densidad de probabilidades "PDF" de dicha señal, la cual tiene una forma gaussiana, parte superior de la Figura 3.2. La función de densidad de probabilidad para una señal de ruido blanco esta determinada por la ecuación 3.3:

$$f(x) = P(x) = \frac{1}{\sigma\sqrt{2\pi}} exp^{-\frac{(x-\mu)^2}{2\sigma^2}}, \qquad -\infty < x < \infty \tag{3.3}$$

dónde, x es una variable aleatoria que tiene una distribución normal, con una media μ y una desviación estándar σ y una varianza σ^2. Dónde, μ y σ^2 son los momentos de primer y segundo orden de la función de densidad de probabilidades, ecuaciones 3.4 y 3.5, respectivamente.

$$\mu = \int_{-\infty}^{\infty} xf(x)dx, \tag{3.4}$$

$$\sigma^2 = \int_{-\infty}^{\infty} (x-\mu)^2 f(x)dx \tag{3.5}$$

La media o valor esperado μ de la variable x es el centro de la función de densidad de probabilidades. La varianza σ^2 es una medida de la dispersión de la variable aleatoria alrededor de la media μ. En el caso particular de la función de densidad de probabilidades gaussiana, la media es también el punto medio en el cual la función de densidad de probabilidades es máxima $(\frac{1}{\sigma\sqrt{2\pi}})$.

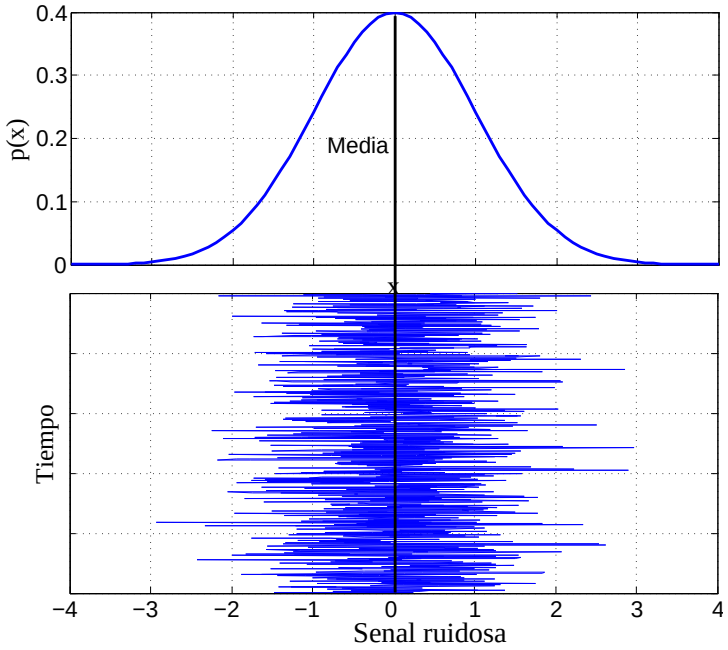


Figura 3.2: Parte inferior: Señal de ruido blanco gaussiano aditivo o ruido térmico, con una media $\mu = 0$ y desviación estándar $\sigma = 1.5$. Parte superior: Función de densidad Probabilística para la señal de la parte inferior de esta Figura.

Las funciones de densidad de probabilidades presentadas en la Figura 3.3 tienen el mismo valor de media $\mu = 0$ con desviación estándar cuadrática o varianza diferente, con la relación $\sigma 1^2 > \sigma 2^2 > \sigma 3^2$, de donde se visualiza que para una mayor varianza, más grande es la dispersión alrededor de la media.

Ruido de disparo

El ruido de disparo en dispositivos electrónicos consiste en el resultado de las fluctuaciones aleatorias de la corriente eléctrica a través de un conductor, causadas por el hecho de que la corriente se transporta en cargas discretas (electrones). Es originado por el movimiento de los electrones o de otras partículas cargadas a través de una unión. Esto no sólo ocurre en las uniones p-n, sino en cualquier conductor, incluso en las situaciones en que la carga no esté bien localizada. Debe distinguirse el ruido de disparo de las fluctuaciones de corriente en equilibrio, las cuales se producen sin aplicar ningún voltaje y sin necesidad de que exista ningún flujo promedio de corriente. Estas fluctuaciones de corriente de equilibrio se conocen como ruido

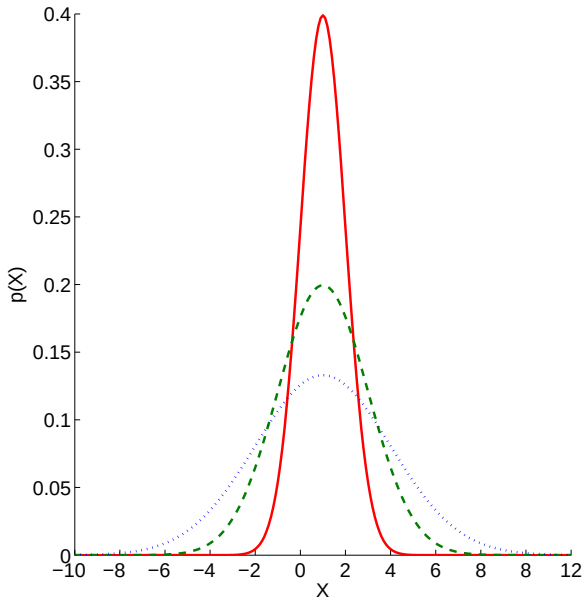


Figura 3.3: Función de densidad probabilística Gausiana para una varianza $\sigma 1 = 3^2$, $\sigma 2 = 2^2$ y $\sigma 3 = 1$, representadas con una línea punteada, con guiones y continua, respectivamente.

de Johnson-Nyquist. El ruido de disparo se puede modelar como un proceso de Poisson y de esta forma los portadores de carga que forman la corriente siguen una distribución de Poisson.

El ruido de disparo es un tipo de ruido que tiene lugar cuando un número finito de particulas que transportan energía, tales como electrones en un circuito electrónico es suficientemente pequeño para dar lugar a la aparición de fluctuaciones estocásticas. El nivel de señal de ruido crece más rápidamente en cuanto mayor es su nivel promedio de corriente eléctrica y por lo tanto, este tipo de ruido se hace más representativo cuando se trabaja con intensidades de corriente bajas.

La densidad de la distribución de Poisson se expresa por la siguiente ecuación:

$$f(k;\lambda) = \frac{e^{-\lambda}\lambda^k}{k!} \tag{3.6}$$

donde:

- k es el número de ocurrencias del evento o fenómeno.

- λ representa el número de veces que se espera que ocurra el fenómeno durante un intervalo dado.
- e es la base de los logaritmos naturales igual a 2,71828...

Por lo tanto, la función f proporciona la probabilidad de que el evento suceda k veces, para una distribución de Poisson. Tres distribuciones de Poisson para una λ de 1, 4 y 10 se muestran en la Figura 3.4.

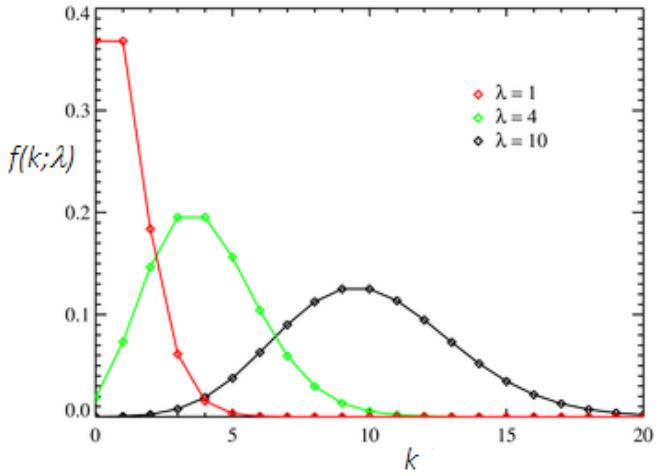


Figura 3.4: Distribución de Poisson. El eje horizontal es el índice k. La función solamente está definida en valores enteros de k. Las líneas que conectan los puntos solo son de ayuda visual y no debe entenderse como una continuidad.

Ruido Flicker

El ruido Flicker o de parpadeo es un tipo de ruido electrónico con un espectro de densidad de potencia dado por $1/f$. Este tipo de ruido se produce en casi todos los dispositivos electrónicos, y puede presentarse con una variedad de otros efectos, tales como dopantes en el canal de un conductor; la generación de ruido de recombinación en un transistor debido a la corriente de base. El ruido Flicker en corriente o voltaje siempre esta relacionado con una corriente continua, ya que es una fluctuación de la resistencia que se transforma a fluctuaciones de voltaje o corriente. En dispositivos electrónicos se manifiesta como un fenómeno de baja frecuencia, ya que a altas frecuencias la magnitud del ruido blanco es mayor y se ven enmascaradas por estas. El ruido Flicker es a menudo caracterizado por la frecuencia de corte fc que se encuentra determinada por la región de baja frecuencia dominada por ruido

Flicker y la región de alta frecuencia dominada por ruido blanco o banda plana. La potencia de voltaje de ruido Flicker en un MOSFET puede ser expresada por $K/(Cox * WLf)$, donde K es la constante que depende del proceso, Cox es la capacitancia del oxido en un dispositivo MOSFET, W y L son el ancho y largo del canal, respectivamente [56].

De los principales tipos de ruido se puede concluir que:

- El ruido térmico está presente en todos los componentes que disipan energía.
- Siempre que haya corriente fluyendo a través de un material no homogéneo, habrá ruido Flicker .
- En un escenario de bajas frecuencias el ruido Flicker es la principal fuente de ruido.
- Los ruidos Flicker y Shot son directamente proporcionales a la corriente, por lo tanto, una condición conveniente para los circuitos es mantener las corrientes de polarización al mínimo.

Los sistemas de cómputo son desarrollados durante un período de tiempo, los cuales pasan por una serie de fases o etapas; la de especificación, la de diseño, la de prototipo, la de implementación y, finalmente, la de instalación. Una avería, un error o un fallo puede ocurrir durante una o más de esas fases. Donde, una avería (fault) se define como un defecto físico que tiene lugar en alguna parte del sistema. Una avería que ocurre durante la etapa de desarrollo puede llegar a ser evidente sólo en una etapa posterior. Las averías se manifiestan en forma de error. Cuando se detecta un error durante la operación de un sistema, puede dar lugar a un fallo (failure). Un sistema se dice que ha fallado, si no puede cumplir con su función. La figura 3.5 muestra un ejemplo que ilustra los tres términos; avería (fault), error (error) y falla (failure).

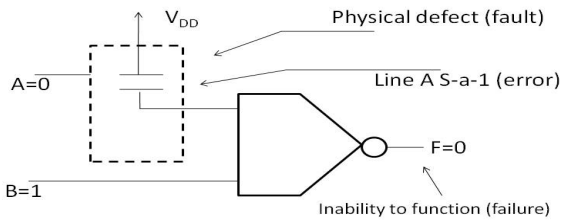


Figura 3.5: Ejemplo de relación entre avería (fault), error y fallo (failure).

Un enfoque general de tolerancia a fallos es el uso de redundancia, el cual coadyuva al correcto funcionamiento de un sistema cuando ocurren fallos específicos. Para enmascarar defectos o reconfigurar un sistema defectuoso se puede implementar redundancia a nivel de hardware, software, información o tiempo [57].

En un diseño de sistemas tolerante a fallos, los diseñadores deben considerar la posible ocurrencia de varios y diferentes tipos de faults, tal como: faults transitorios, faults intermitentes, faults permanentes, faults lógicos y faults indeterminados. Los faults transitorios a menudo son perturbaciones externas, las cuales tienen una longitud de tiempo finita no periódicos. Los faults intermitentes ocurren periódicamente y típicamente resultan de la operación de un dispositivo inestable. Los faults permanentes son perpetuos y pueden ser causados por daños físicos en los procesos de fabricación o errores de diseño. Los faults lógicos ocurren cuando las entradas o salidas de las puertas lógicas están en stuck-at-0 o stuck-at-1. Los faults indeterminados ocurren cuando las entradas o salidas de las puertas lógicas fluctúan entre un 0 lógico y un 1 lógico [58]. Un sistema puede operar correctamente en presencia de los defectos antes mencionados, si se incorpora en el sistema la o las formas apropiadas de redundancia [58]. Dos de los principales enfoques de tolerancia a faults son redundancia estática y redundancia dinámica. Redundancia estática es el uso de replicas de componentes y/o dispositivos de manera que los faults podrían ser enmascarados, tal como la técnica basada en los campos aleatorios de Markov [24]. Redundancia dinámica es la reorganización de un sistema, de modo que las funciones de una unidad defectuosa se transfieren a otras unidades funcionales [59], por ejemplo, la técnica titulada RAZOR [60, 61, 62], que a nivel componente lógico utiliza un flip-flop principal y un latch sombra, donde el latch sombra utiliza un reloj retrasado respecto al reloj que utiliza el flip-flop y así asegura que el latch sombra tenga el dato correcto, el cual, si y solo si, difiere del dato almacenado temporalmente en el flip-flop principal, el dato que tiene el flip-flop sombra es insertado en la cadena de datos por medio de un mecanismo de retroalimentación.

Existen cuatro tipos específicos de redundancia: redundancia de información, redundancia en tiempo, redundancia de software, y redundancia de hardware, donde:

- Redundancia de información: es el uso de detección de errores o códigos de corrección de errores para la representación de información.
- Redundancia de tiempo: es la repetición de las operaciones del sistema a fin de que los fallos transitorios pueden ser enmascarados.

- Redundancia de software: es la inclusión de varios programas alternativos para las operaciones del sistema a fin de que fallos de software (errores de diseño) sean tolerados.
- Redundancia de hardware: es la inclusión de múltiples copias de los componentes críticos de manera que los fallos intermitentes y permanentes se puede tolerar.

3.1 Error en un nodo digital causado por ruido

En este trabajo las señales de voltaje en un nodo digital, son modelados por medio de dos niveles lógicos 0 y 1, con ruido Gaussiano blanco aditivo (*Additive White Gaussian Noise* AWGN) no correlado, con una media μ, una desviación estándar σ y una varianza σ^2. La Figura 3.6 muestra la distribución de probabilidades de voltajes en un nodo digital ruidoso, dónde este puede ser descrito como la unión de dos distribuciones Gaussianas centradas alrededor del 0-lógico y 1-lógico (0 y V_{DD}, respectivamente). Un circuito digital presenta un error cuando un valor lógico es mal interpretado, es decir, cuando un 1-lógico es interpretado como un 0-lógico y viceversa, esto es resaltado por las áreas sombreadas en la Figura 3.6. De dónde las áreas sombreadas en la Figura 3.6 representan la probabilidad de que un circuito presente un error en sus salidas. Las respectivas probabilidades mal interpretadas o de error en los estados lógicos 1 y 0, son denotadas por las variables P_{e1} y P_{e0}, respectivamente. Ellas son obtenidas integrando las funciones de densidad de probabilidades (PDF) de un 1-lógico y un 0-lógico, más allá de los valores lógicos ($\frac{V_{DD}}{2}$), el cual corresponde al voltaje de umbral entre un 1-lógico y un 0-lógico, ecuaciones (3.7) y (3.8). Por simplicidad se consideran umbrales de ruido lógicos nulos ya que esta consideración no afecta el resultado, ya que en caso estricto este valor forma parte de los niveles 1-lógico y 0-lógico.

$$P_{e1}(x) = \int_{-\infty}^{\frac{V_{DD}}{2}} \frac{1}{\sigma\sqrt{2\pi}} exp^{-\frac{(x-\mu)^2}{2\sigma^2}}, \qquad -\infty < x < \infty \tag{3.7}$$

$$P_{e0}(x) = \int_{\frac{V_{DD}}{2}}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} exp^{-\frac{(x-\mu)^2}{2\sigma^2}}, \qquad -\infty < x < \infty \tag{3.8}$$

Asumiendo la misma densidad de probabilidades para un 1-lógico y un 0-lógico ($\sigma_1 = \sigma_0$), la probabilidad de que un valor de voltaje en un nodo ruidoso sea incorrecto es P_e, y esta dado por la ecuación (3.9).

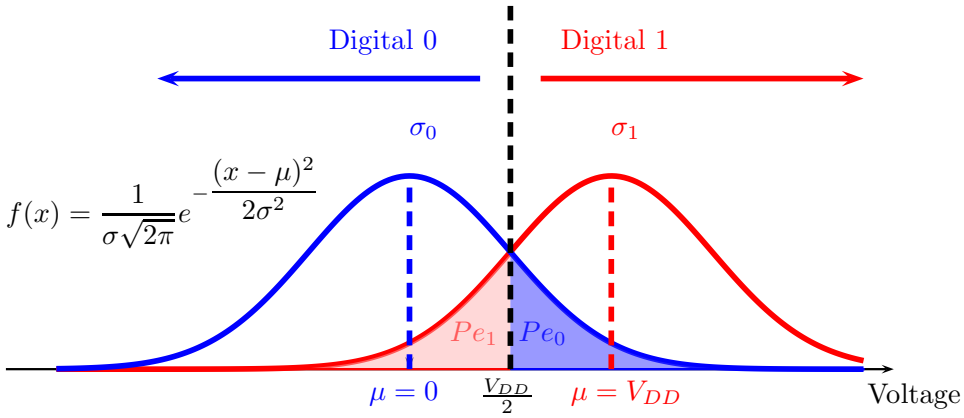


Figura 3.6: Función de densidad de probabilidades para valores digitales en un nodo digital bajo la influencia de ruido blanco gaussiano aditivo, con voltajes esperados de $\mu = 0$ y $\mu = V_{DD}$ con el mismo valor de desviación estándar para σ_0 y σ_1.

$$P_e = \frac{P_{e_0} + P_{e_1}}{2} \tag{3.9}$$

El valor de P_e en términos de los parámetros de una función de distribución Gaussiana puede ser expresada como:

$$\begin{aligned} P_e &= \frac{\int_{\frac{V_{DD}}{2}}^{\infty} f(\epsilon)\, \mathrm{d}\epsilon + \int_{-\infty}^{\frac{V_{DD}}{2}} f(\epsilon)\, \mathrm{d}\epsilon}{2} \\ &= \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{V_{DD}/2}{\sigma\sqrt{2}}\right)\right) \end{aligned} \tag{3.10}$$

dónde $f(\epsilon)$ es la función de distribución Gaussiana [26].

Por lo tanto, la probabilidad de que el valor lógico en un nodo ruidoso sea correcto es:

$$P_c = 1 - P_e \tag{3.11}$$

3.2 Técnica de redundancia de puertos (PR)

En esta sección se analiza como mejorar la fiabilidad de un módulo convencional replicando el número de puertos del mismo módulo, el cual en este trabajo es nombrado como Redundancia de Puertos (PR) Figura 3.7. Donde la Figura 3.7(c) presenta como un módulo puede tener $(n-1)$ puertos replicados, siendo n el numero total de puertos que tiene un módulo. La lógica convencional diferencial corresponde al caso particular de $(n = 2)$, Figura 3.7(b). Los puertos replicados se pueden realizar usando el mismo puerto (verdadero) o el negado (complemento) del puerto original. En este trabajo se hace la suposición que cada puerto redundante es generado usando diferente driver, por lo tanto el ruido puede considerarse estadísticamente independiente en cada uno de ellos.

En general sin importar la forma de replicar los puertos, las entradas de los puertos replicados pueden exhibir tres diferentes escenarios para sus valores de voltaje, nombrados correcto, erróneo y discrepante, los cuales cuentan con probabilidades que pueden ser descritas y calculadas como sigue:

1. *Escenario con valores correctos (CS)*: Cuando todas los valores lógicos de entrada son interpretados correctamente (no hay error en ninguna entrada). La probabilidad de este escenario esta definido por P_{cn}, cuando n puertos son considerados ecuación (3.12).

$$P_{cn} = (1 - P_e)^n \tag{3.12}$$

2. *Escenario con valores de voltaje erróneos (ES)*: Es cuando los valores de voltaje de todos los puertos son mal interpretados al mismo tiempo, por ejemplo, cuando un 1-lógico es interpretado como un 0-lógico, o viceversa. En este caso, los valores interpretados son aparentemente coherentes. La probabilidad de este escenario esta dado por P_{en}, ecuación (3.13).

$$P_{en} = P_e^n \tag{3.13}$$

3. *escenario con valores de voltaje discrepantes (DS)*: Uno o varios valores de voltaje de los puertos replicados (no todos), tienen valores de voltaje erróneos los cuales pueden causar una incorrecta interpretación. La probabilidad de que esto ocurra esta determinado por P_{dn} y dada por la ecuación (3.14).

$$P_{dn} = \sum_{k=1}^{n-1} \frac{n!}{k!(n-k)!}(1-P_e)^{n-k}P_e^k \tag{3.14}$$

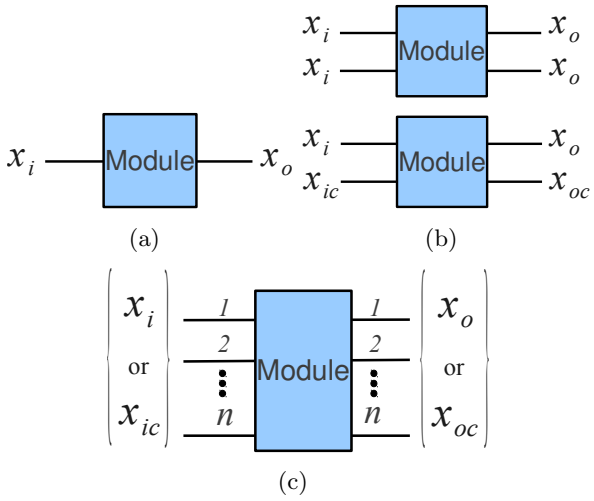


Figura 3.7: Redundacia de puertos. (a) Puerta lógica simple con un puerto de entrada y un puerto de salida, (b) Caso particular para $n = 2$, (c) Puerta lógica simple con $(n-1)$ puertos replicados. Cabe hacer mención que el número total de puertos después de aplicar una redundancia de puertos es n.

La suma de los tres posibles escenarios es igual a uno, como puede observarse en la ecuación 3.15.

$$P_{cn} + P_{en} + P_{dn} =$$

$$= \sum_{k=0}^{n} \frac{n!}{k!(n-k)!}(1-P_e)^{n-k}P_e^k = 1 \tag{3.15}$$

Para PR-N donde N es el número de puertos replicados, debe entenderse que el número total de puertos es $n = N+1$. Consecuentemente una puerta lógica NOT con PR-1 significa que la puerta lógica NOT tiene el doble de puertos, en otras palabras la puerta lógica NOT tiene dos entradas y dos salidas.

3.3 Escenarios probabilísticos para una puerta lógica NOT con PR-1

En el caso de replicar el número de puertos de un puerta lógica NOT mediante la redundancia de puertos con PR-1, existen tres posibles escenarios paralelamente a las mencionados: llamados correcto, erróneo y discrepante en concordancia con los mencionados en la sección anterior.

La replica de puertos puede realizarse de tres maneras: en el primer caso con sus mismos o verdaderos valores lógicos, en el segundo caso con sus negados o complementarios y finalmente el tercer caso con una combinación entre verdaderos y complementarios. Sin importar la manera de replicar los puertos de la puerta lógica NOT existen tres escenarios y sus probabilidades se describen a continuación:

1. *CS para* $n = 2$: Ambos valores de voltaje son correctos, Figura 3.8(a), la probabilidad de ocurrencia es descrita por la ecuación (3.16).

$$P_{c2} = (1 - P_e)(1 - P_e) = (1 - P_e)^2 \tag{3.16}$$

2. *ES para* $n = 2$: Ambos valores de voltaje son incorrectos, Figura 3.8(b), aparentemente este caso es coherente para el sistema por que ambos valores son correctos, sin embargo es erróneo. La probabilidad de ocurrencia es descrita por la ecuación (3.17).

$$P_{e2} = P_e P_e = P_e^2 \tag{3.17}$$

3. *DS para* $n = 2$: Sólo uno de ambos voltajes es incorrecto, Figura 3.8(c), la ecuación (3.18) describe la probabilidad de ocurrencia.

$$P_{d2} = 2P_e(1 - P_e) \tag{3.18}$$

3.4 Reduciendo la probabilidad de error usando redundancia de puertos (PR-N)

La probabilidad de error (P_{en}) obedece a una relación logarítmica respecto al número de puertos replicados, tal como lo expresa la ecuación (3.13). La Figura 3.9 presenta los resultados obtenidos de la probabilidad de error para una puerta lógica NOT con redundancias de 0 a 9, para un SNR dentro del

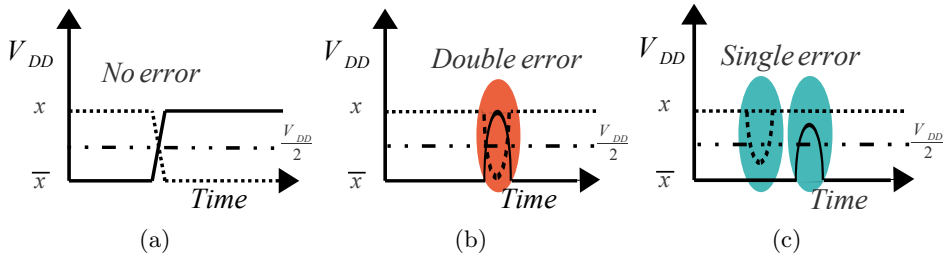


Figura 3.8: Posibles escenarios para una puerta lógica con un puerto complementario replicados (x, $\overline{x}$), (a) Ambos valores lógicos son correctos (P_{c2}), (b) Ambos valores lógicos son incorrectos (P_{e2}) por causa de un ruido transitorio y, (c) Sólo un valor lógico de ambos es incorrecto (P_{d2}).

rango de -3 to 16 dB. Tal como se esperaba, si el número de puertos replicados se incrementa la probabilidad de error de la puerta lógica NOT se reduce. Para el caso especifico de PR-0 (puerta convencional sin redundancia) con un SNR=-3dB ($\sigma = 0.5$ (Vrms) y $V_{DD} = 0.5$ (V)), la probabilidad de error es 35%. En cambio, para una redundancia de PR-1 la probabilidad de error esta alrededor del 12% que significa una reducción del 65% de la probabilidad de error respecto a PR-0. Cuando PR es incrementada a 2, 3 y 4 veces, las correspondientes probabilidades de error se reducen cada vez más, obteniendo los valores de 11.95%, 4.13%, 1.41%, 4.91e-3%, respectivamente. Cuando PR va más allá de 4 veces se observa un decremento insignificante de la probabilidad de error, con la desventaja de que se incrementan los requerimientos de hardware para implementar un modulo con la misma funcionalidad y prestaciones deseadas. El coste en hardware respecto al incremento de la redundancia de PR en un circuito digital es inversamente proporcional. Por lo tanto y con base al análisis anterior, una redundancia de PR-1 es considerado en este trabajo, ya que es un buen balance entre fiabilidad y *overhead*. Para un SNR menor a -3dB, la tendencia de la probabilidad de error para PR-0, PR-1, ..., PR-9, es totalmente exponencial y converge a $\frac{1}{2^n}$.

Si se obtiene la probabilidad de error de PR-0, PR-1, ..., PR-9 para un SNR desde -100 dB hasta 25 dB, para el caso de -100dB existe un extremadamente alto ruido y ningún sistema trabaja, porque la potencia del ruido del sistema es mucho mayor que la potencia de la señal, de hecho es un escenario totalmente probabilístico, ya que cualquier dispositivo lógico deja de realizar la operación lógica para la cual fue diseñado, y sus salidas pueden tomar cualquier valor lógico (1-lógico o 0-lógico).

Como se esperaba la probabilidad de error P_{en} es $1/(2^n)$, lo cual ocurre para valores de voltaje completamente aleatorios que se presentan para un

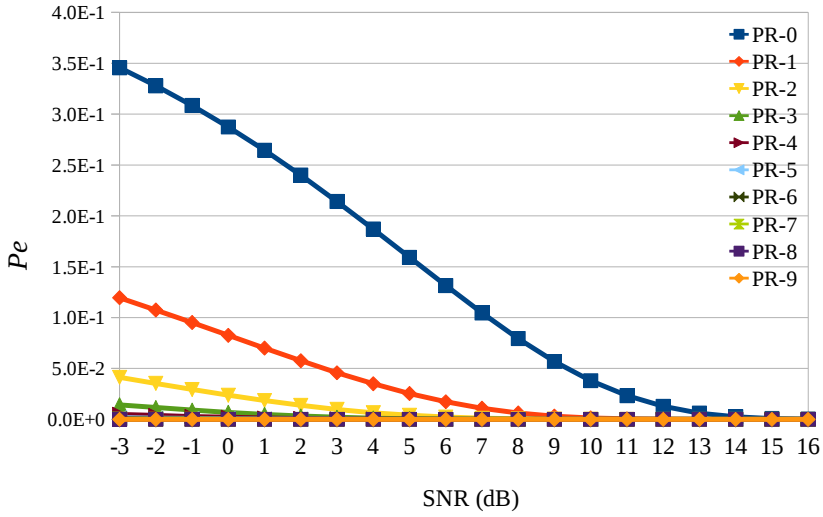


Figura 3.9: Probabilidad de error para una puerta simple con una redundancia de puertos de 0-9. Cabe señalar que PR-0 corresponde al caso de un módulo sin redundancia de puertos.

$SNR < -50$ (dB), dónde n es el número total de puertos. En este escenario extremadamente ruidoso o totalmente estocástico, la máxima probabilidad de error para PR-0 es $\frac{1}{2}$, $\frac{1}{4}$ para el caso de PR-1, $\frac{1}{8}$ para PR-2, y así sucesivamente hasta n, tal como puede observarse en la Figura 3.10.

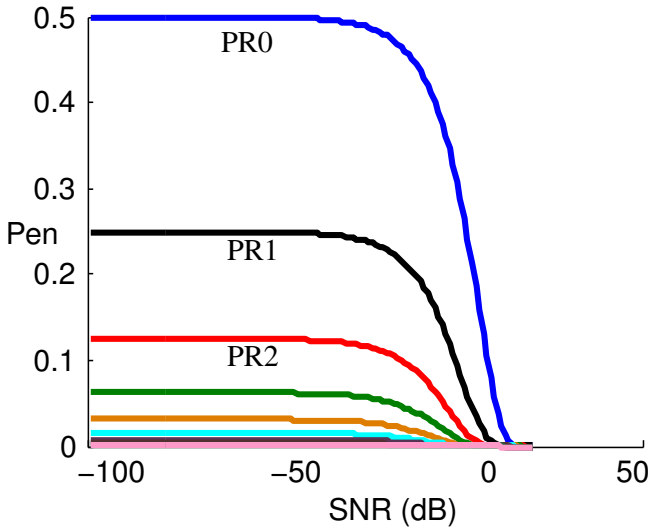


Figura 3.10: Probabilidad de error para una puerta simple con redundancia de puertos para PR0-PR9, para un escenario extremadamente ruidoso representado por un barrido de SNR de -100dB a 25dB.

3.5 Conclusiones

En tecnologías de dimensiones nanométricas debido a la reducción de los tamaños característicos, más pequeños voltajes de alimentación (márgenes de ruido más pequeños) y densidad de integración más grande [6], la inmunidad al ruido presenta una seria dificultad para la fiabilidad de estas tecnologías.

Los principales tipos de ruido en semiconductores son ruido térmico, ruido de disparo y ruido flicker y se puede concluir que:

- El ruido térmico está presente en todos los componentes que disipan potencia.
- Habrá ruido flicker siempre que haya corriente fluyendo a través de un material no homogéneo.
- En bajas frecuencias el ruido Flicker es la principal fuente de ruido.
- Los ruidos Flicker y Shot son proporcionales a la corriente, por lo tanto, hay que mantener las corrientes de polarización al mínimo.

La distribución de probabilidades de voltajes en un nodo digital ruidoso, puede ser descrito como la unión de dos distribuciones Gaussianas centradas alrededor del 0-lógico y 1-lógico (0 y V_{DD}, respectivamente).

Un circuito digital presenta un error cuando un valor lógico es mal interpretado, es decir, cuando un 1-lógico es interpretado como un 0-lógico y viceversa, esto es mostrado en la Figura 3.6 y esta definido por la probabilidad de error P_e de la ecuación 3.10.

Una posible forma de reducir la probabilidad de error de un nodo digital ruidoso es mediante la redundancia de puertos (PR). De donde si, el número de puertos replicados se incrementa, la probabilidad de error se reduce.

Tomando como base la replica de puertos en una puerta lógica NOT, al replicar sus puertos de entrada y salida, para el caso especifico de PR-0 (puerta convencional sin redundancia) con un SNR=-3dB ($\sigma = 0.5$ (Vrms) y $V_{DD} = 0.5$ (V)), la probabilidad de error es 35%.

Para el caso de una redundancia de PR-1, la probabilidad de error está alrededor del 12%, lo que es equivalente a una reducción del 65% de la probabilidad de error, respecto a PR-0.

Cuando PR se incrementa a 2, 3 y 4 veces, las correspondientes probabilidades de error se reducen cada vez, obteniendo los valores de 11.95%, 4.13%, 1.41%, 4.91e-3%, respectivamente.

Cuando PR va más allá de 4 veces se observa un decremento insignificante con la desventaja del incremento en los requerimientos de hardware para implementar el circuito con la misma funcionalidad y prestaciones deseadas.

El coste en hardware es inversamente proporcional al incremento de la redundancia de PR en un circuito digital.

Por lo tanto y con base al análisis anterior, una redundancia de PR-1 es considerado en este trabajo, ya que es un buen balance entre fiabilidad y *overhead*.

Capítulo 4

Propuesta de metodología y Objetivos de la tesis

4.1 Propuesta de Metodología

La metodología propuesta en esta tesis está basada en el uso de redundancia de puertos en todos los elementos de procesamiento lógico. En otras palabras la redundancia de puertos, equivalente a que cualquier elemento de procesamiento lógico se repliquen n veces sus puertos tanto de entrada como sus puertos de salida.

De acuerdo al análisis realizado en la sección 3.4, se demostró que una relación adecuada entre redundancia, complejidad y penalización en hardware, es que todos los puertos de los circuitos hagan uso de una redundancia de puertos PR-1. Esta redundancia de puertos PR-1 se ve reflejada en que el número de puertos resultantes sea duplicado. Esto es, si se tiene un circuito sin redundancia con dos entradas a y b y una salida c, después de aplicar una redundancia de puertos PR-1 se obtienen un circuito con cuatro puertos de entrada y dos puertos de salida. La redundancia de puertos se puede realizar de varias maneras, que puede ser mediante la replicación de los mismos puertos o replicandolos con sus puertos complementarios. Respecto a este escenario en la sección 3.4, se trata de una replica de puertos mediante el mismo puerto lógico o mediante su puerto complemento, así mismo se demostró que la manera más adecuada en materia de fiabilidad es por medio de su puerto complemento.

La metodología de diseño que se usa en esta tesis, consiste en el análisis

de la coherencia entre las señales de entrada de los puertos verdadero y complementario, en dónde coherencia debe entenderse como señales lógicas de entrada complementarias (señal lógica verdadera y su complemento lógico). Cada bloque lógico verifica la coherencia de todas las señales de los puertos complementarios de entrada. En caso de que las entradas presenten coherencia en sus respectivos bloques, módulos o elementos lógicos, entonces el módulo realiza la operación lógica normal para la cual fue diseñado. En caso contrario los datos de salida permanecen en su estado lógico correcto previo. Cada bloque, módulo o elemento lógico debe de entenderse como puerta lógica, circuito combinacional, circuito secuencial o sistema.

En caso de que al menos una de las entradas con PR-1 presente incoherencia entre si, entonces la o las salidas de los elementos lógicos permanecen en sus valores correctos previos, de esta manera se evita que se propaguen valores incorrectos de las entradas a las salidas. Este concepto lógico es llamado Lógica Tortuga (Turtle Logic: TL) debido a que imita el comportamiento de las tortugas en la naturaleza. Esto es cuando las tortugas se encuentran en una situación de peligro, ellas permanecen en un estado de auto protección. Cuando la amenaza ha desaparecido las tortugas continúan con su vida normal. Siguiendo esta analogía cuando existe ruido que provoque que las señales redundantes complementarias presenten valores lógicos incoherentes, entonces el dispositivo mantiene sus valores lógicos de salida previos, y cuando el ruido tiene una magnitud tal que no provoque una incoherencia en algunas de las entradas complementarias, entonces los dispositivos tortuga continúan con su operación lógica normal para la cual fueron diseñados.

4.2 Objetivos

Obtener una nueva metodología para el diseño de circuitos integrados digitales robustos en escenarios con ruido extremo y con bajo voltaje de alimentación.

Los objetivos particulares de acuerdo a la propuesta de tesis que coadyuvan al logro del objetivo principal son:

1. Investigar los escenarios de las nuevas y/o futuras tecnología en términos de:
    - Voltaje de alimentación.
    - Ruido que afecta a un circuito digital.

- Tolerancia de circuitos digitales frente a fallos ("Soft-faults").

2. Analizar las metodologías existentes para el Diseño de Circuitos Integrados Digitales que mejoren la fiabilidad de los mismos en escenarios con grandes cantidades de ruido y en escenarios de bajo voltaje de alimentación, por ejemplo:
   - N-Tuple Modular Redundancy.
   - C-Element.
   - Técnicas probabilísticas.
     - Markov Random Field.
3. Proponer una metodología de diseño de circuitos integrados digitales que permita mejorar la fiabilidad de los mismos en escenarios con magnitudes altas de ruido y bajo voltaje de alimentación.
4. Diseñar una topología para los elementos NOT, BUFFER, AND, NAND, OR, NOR, XOR, XNOR.
5. Analizar y seleccionar los parámetros de evaluación de la propuesta en términos de:
   - Voltaje de alimentación (Voltaje de umbral para transistores futuros y umbral para los niveles lógicos).
   - Ruido blanco en las señales de entrada, salida y ambas; ruido en elementos internos: y márgenes de ruido en circuitos digitales.
   - Tolerancia frente a fallos.
   - Consumo de potencia.
   - Consumo de área.
   - Velocidad.
6. Diseñar una arquitectura para la propuesta realizada.
   - Handshake.
   - Pipeline.
   - Multiplicador como medio de prueba.
7. Comparar la propuesta realizada contra propuestas previas
   - Tuple Modular Redundancy.
   - C-Element.
   - Markov Random Field.
8. Verificación
   - Multiplicador como medio de prueba.
9. Publicación de resultados.

Capítulo 5

Concepto de Lógica Tortuga, Aplicación en puertas lógicas y circuitos combinacionales

A medida que se ha avanzando el diseño CMOS a un escenario nanométrico el número de errores que tiene que enfrentar un circuito lógico se ha incrementado. En este escenario el ruido térmico llega a ser más relevante, por lo tanto, la relación de señal a ruido cada día es más pequeña y grandes cantidades de errores se pueden hacer presentes. Esto se ve reflejado en las prestaciones de los circuitos. Debido a la naturaleza aleatoria y dinámica de los fallos o errores, una aproximación probabilística es la más apropiada para el manejo de este tipo de escenarios.

5.1 Puerta lógica TL-NOT

Para mostrar la operación del diseño de la lógica Tortuga, se utiliza una puerta lógica NOT (TL-NOT) con redundancia PR-1 usando el valor lógico complementario para la replica de puertos, la cual presenta tres posibles modos de operación:

- Primer modo de operación: Cuando los puertos de entrada presentan valores complementarios ($x_{it} \neq x_{ic}$), entonces, la puerta lógica TL-NOT asume las entradas como valores correctos y forza los puertos de salida complementarios a los valores apropiados de acuerdo a la

función lógica NOT.

- Segundo modo de operación: Las entradas tienen el mismo valor ($x_{it} = x_{ic}$), entonces, las entradas no son consistentes entre ellas mismas y por lo tanto, la puerta lógica TL-NOT identifica estos valores como incoherentes o erróneas, haciendo que los puertos de salida del sistema mantengas sus valores correctos previos.
- Tercer modo de operación: En caso de error en ambas variables de entrada (doble error). La técnica tortuga asume que las variables de entrada son validas y consecuentemente puede generar un error en sus salidas redundantes. Sin embargo, la probabilidad de que ocurran eventos simultáneos y complementarios es muy pequeña como lo demuestra la ecuación 3.13, Figura 3.9. Dado que tendría que ocurrir que mientas la energía en una de las dos señales redundantes crece, la energía de la otra señal decrece.

Un posible diseño para la implementación de la función lógica TL-NOT a nivel puerta como a nivel transistor se muestra en la Figura 5.1 incisos (b) y (c), respectivamente. Si se toma el símbolo de una puerta lógica y se le aplica una redundancia PR-1, entonces se obtiene una puerta lógica NOT con dos entradas y dos salidas como el que se presenta en la Figura 5.1(a). La forma de como se obtienen estos circuitos es utilizando todos los posibles valores de entrada para dos entradas lógicas y complementarias.

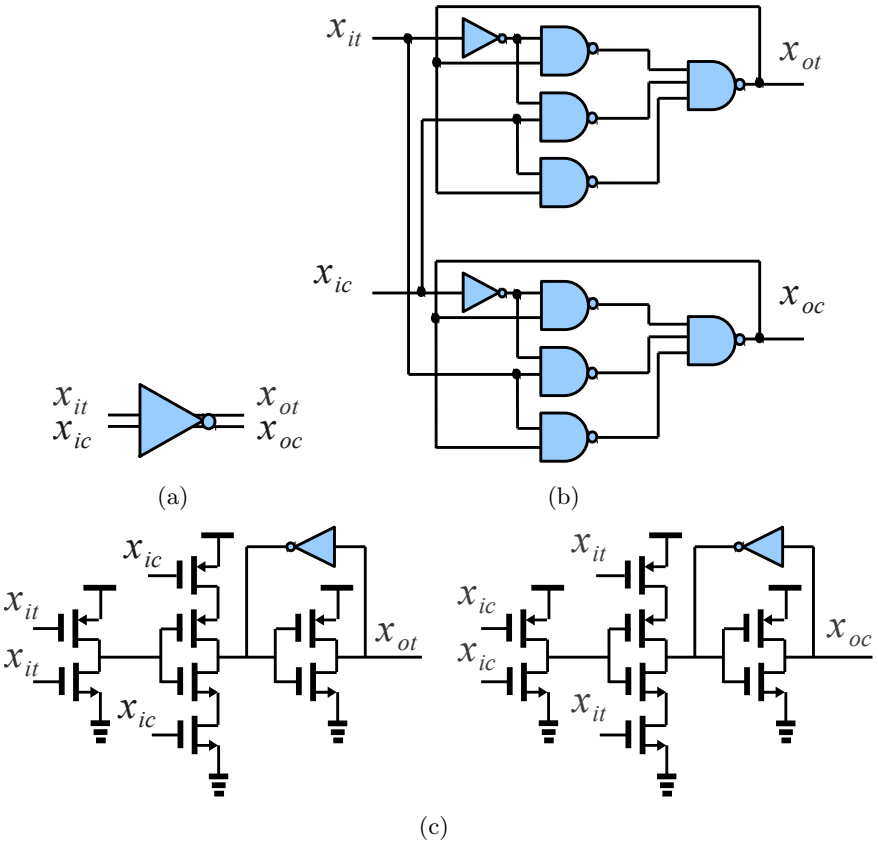


Figura 5.1: Puerta lógica TL-NOT. (a) Símbolo (b) diseño a nivel puerta, y (c) diseño a nivel transistor.

5.2 Puerta lógica TL-NAND2

Cualquier puerta lógica puede ser implementada por medio de los principios de la lógica Tortuga (TL) presentados en la sección anterior.

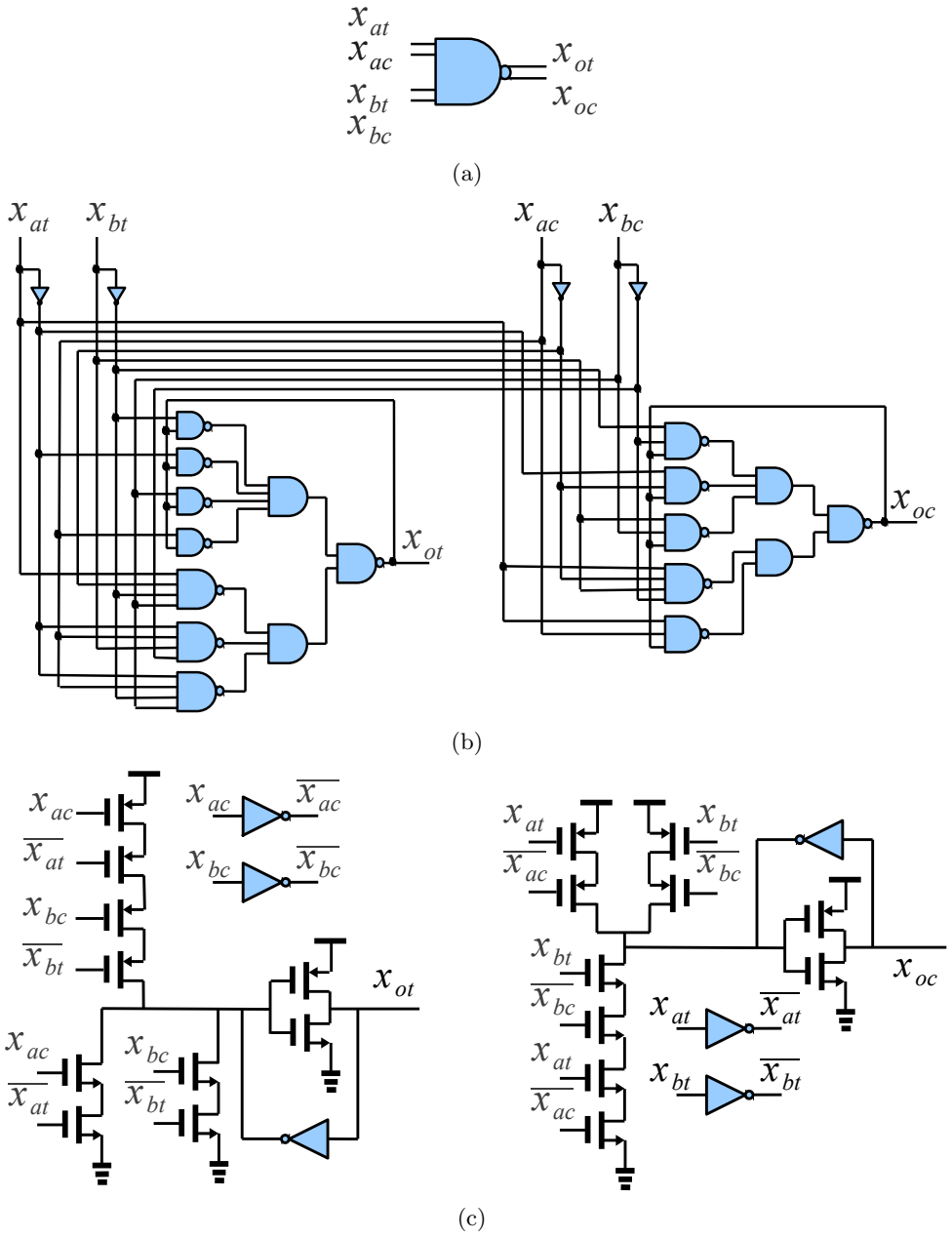


Figura 5.2: Puerta lógica TL-NAND2. (a) Símbolo, (b) Vista esquemática de un posible diseño usando puerta lógicas estándar y (c) Diseño usando transistores.

Un puerta lógica NAND es presentada como ejemplo en esta sección. Se presentan dos posibles formas de implementar una puerta lógica TL-NAND2; la primera es usando puertas lógicas convencionales y la segunda es

usando transistores CMOS, Figura 5.2. Considerando una operación lógica NAND de dos variables de entradas a y b. Se denominan las componentes verdaderas como x_{at} y x_{bt}, y sus componentes redundantes complementarios como x_{ac} y x_{bc}, respectivamente. La puerta lógica TL-NAND2 tiene una descripción similar a la puerta lógica TL-NOT, dónde existen dos posibles modos de operación:

- Primer modo de operación: Cuándo los puertos de entrada de la puerta lógica TL-NAND2 tienen valores lógicos complementarios ($x_{at} \neq x_{ac}$) y ($x_{bt} \neq x_{bc}$), entonces, la puerta lógica TL-NAND2 identifica a los valores de entrada como correctos y fuerza a los puertos de salida a tomar los valores apropiados de acuerdo a la función lógica NAND. Esto significa que $x_{ot} = (\overline{x_{at}.x_{bt}})$ y $x_{oc} = (\overline{x_{ac}.x_{bc}})$.
- Segundo modo de operación: Cuándo los valores de voltaje de las entradas tienen el mismo valor ($x_{at} = x_{ac}$) o ($x_{bt} = x_{bc}$), la puerta lógica TL-NAND2 identifica estos valores no coherentes como un error, esto porque las entradas no son consistentes entre ellas. Por lo tanto, los voltajes de los puertos de salida de la puerta lógica TL-NAND2 son forzados a mantener sus valores previos correctos.

Cualquier función lógica o compleja puede ser implementada siguiendo la metodología de diseño de la lógica TL. En orden a construir las puertas básicas que permiten implementar cualquier circuito combinacional sección 5.7, se diseñaron las puertas lógicas TL-NOR2 y la puerta lógica TL-XOR2, Figuras 5.3 y 5.4, respectivamente.

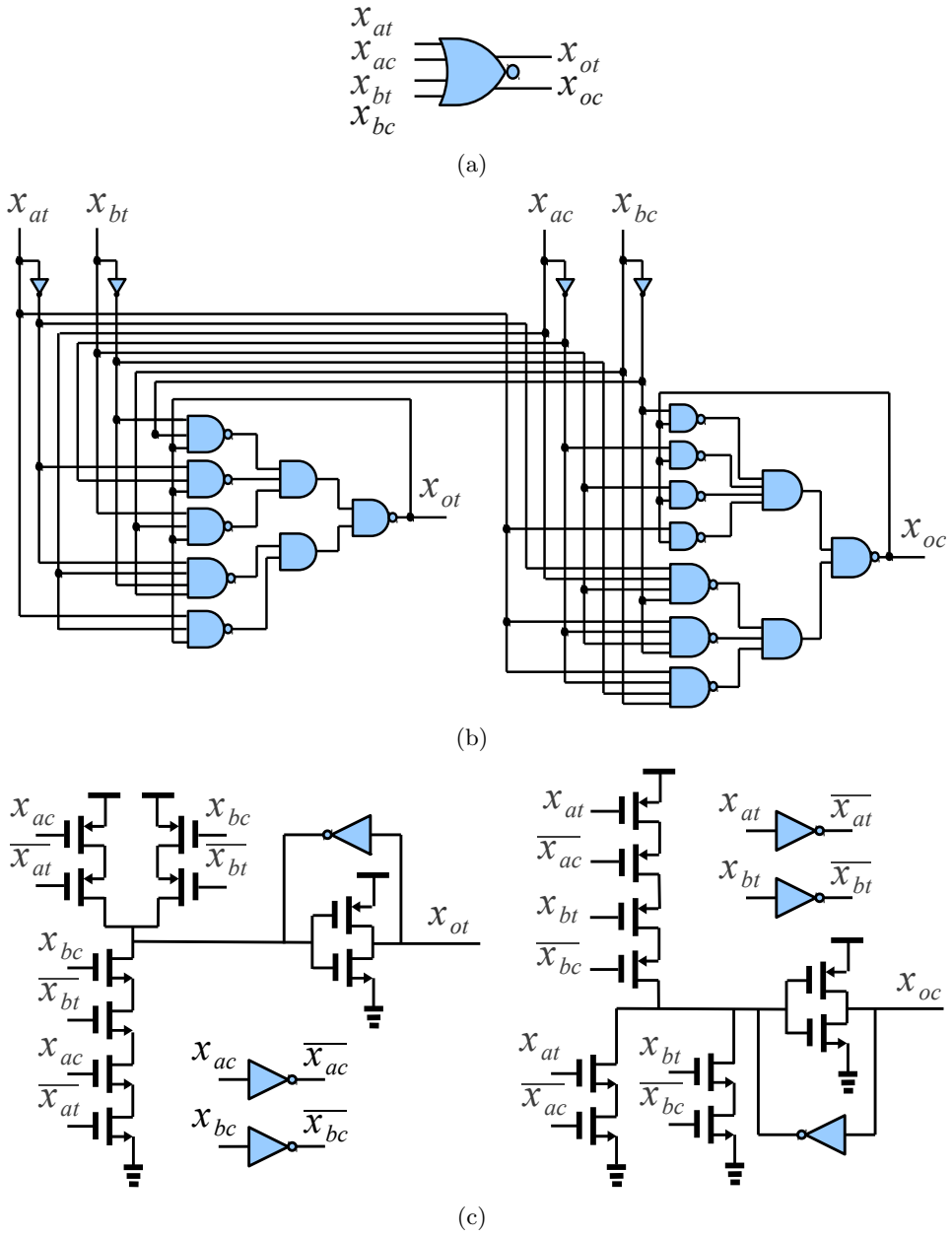


Figura 5.3: Puerta lógica TL-NOR2. (a) Símbolo, (b) Vista esquemática de un posible diseño usando puerta lógicas estándar y (c) Diseño usando transistores.

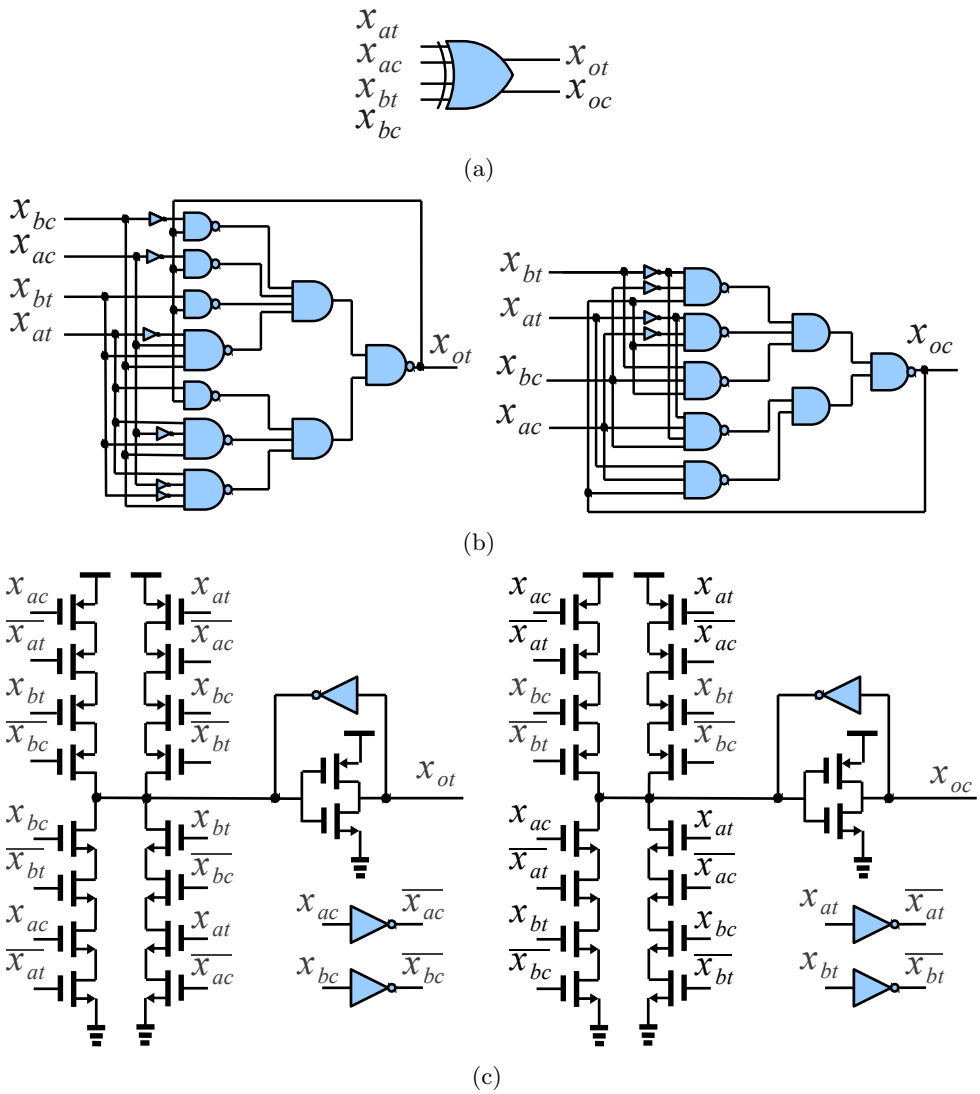


Figura 5.4: Puerta lógica TL-XOR2. (a) Símbolo, (b) Vista esquemática de un posible diseño usando puerta lógicas estándar y (c) Diseño usando transistores.

Cuando un sistema es implementado de acuerdo a un sistema TL hay un compromiso entre un diseño a nivel puerta y un diseño a nivel transistor. El diseño a nivel puerta requiere mucho menos tiempo debido a que los elementos con los cuales se implementa el diseño, son las puertas normales que se encuentran en la biblioteca de componentes de la estructura de diseño CAD. La implementación a nivel transistor exhibirá prestaciones superiores y menor costo en silicio, sin embargo este diseño requiere un detallado dimensionamiento de los transistores.

Cabe resaltar que de acuerdo a lo analizado hasta este punto, se determina que todas las implementación TL, tendrán un coste en hardware (overhead) debido a la redundancia de puertos.

5.3 Penalizaciones de la Lógica Tortuga

La penalización en hardware y prestaciones para TL en comparación con otras técnicas de diseño es un inconveniente importante, por lo tanto, TL esta orientado a tecnologías futuras, dónde la densidad de componentes presumiblemente será mucho mayor que las conocidas hasta ahora. Sin embargo, incluso en el caso de hoy en día las tecnologías convencionales MOSFET, la lógica TL propuesta puede construir alternativas tolerantes a ruido muy grande, obteniendo beneficios de TL que se mostrarán en las siguientes secciones. La tabla 5.1 muestra el costo en hardware introducido por una implementación a nivel transistor respecto a las lógicas convencional CMOS así como a la lógica basada en la teoría de Markov (MRF) tomada la última de [63]. Como puede ser visto el coste en hardware para TL comparado contra CMOS estándar está en un orden de 10, pero en comparación con MRF el coste en hardware de TL es similar o incluso menor. La Tabla 5.2 muestra la comparación de TL respecto a la lógica convencional para las prestaciones de retardo y energía. La energía total consumida ha sido medida para un tiempo de $4ns$. El retardo es el promedio de los retardos medidos para un flanco de subida y uno de bajada para todas las transiciones de salida. El incremento de retardo generado por la estrategia de TL se encuentra entre 1.1X y 2X, mientras que la energía se encuentra entre 2X y 5X veces. Estos resultados pueden ser extrapolados a sistemas complejos de miles o millones de puertas debido a que el impacto de la transformación a TL es lineal. Es importante resaltar que los resultados de costo en hardware y prestaciones para TL presentadas en esta sección corresponden a un diseño utilizando puertas básicas y por lo tanto puede ser mejorado por una implementación a nivel transistor.

Tabla 5.1: Comparativa del coste en hardware para los elementos lógicos de las lógicas CMOS convencional, Markov Random field (MRF) y la Lógica Tortuga (TL).

Puerta	Lógica	# Entradas	# Salidas	Trans.	Overhead
NOT	Conv.	1	1	2	1.0
	MRF	2	2	20	10.0
	TL	2	2	20	10.0
NAND	Conv.	2	1	4	1.0
	MRF	4	2	60	15.0
	TL	4	2	32	8.0
NOR	Conv.	2	1	4	1.0
	MRF	4	2	58	14.5
	TL	4	2	32	8.0
XOR	Conv.	2	1	22	1.0
	MRF	4	2	60	2.7
	TL	4	2	48	2.0

Tabla 5.2: Comparativa de retardo y energía para los elementos lógicos de las lógicas CMOS convencional y la Lógica Tortuga (TL), implementados a nivel transistor mediante el uso de una tecnología de 90nm.

Puerta	Lógica	Retardo (ps)	Energía (fJ)
NOT	Conv.	0.152	2.37
	TL	0.648	3.89
BUFFER	Conv.	0.403	5.24
	TL	0.648	3.89
NAND	Conv.	0.328	2.63
	TL	0.853	7.34
AND	Conv.	0.479	5.00
	TL	0.736	7.34
NOR	Conv.	0.256	4.51
	TL	0.587	7.24
OR	Conv.	0.739	6.87
	TL	0.823	7.24

5.4 Resultados de simulación para una implementación basada en puertas lógicas

Con el fin de comprobar el mejoramiento de la fiabilidad de un circuito por medio de la Lógica Tortuga (TL) se implementa una puerta lógica NOT con tres metodologías diferentes: estándar CMOS , MRF [63] y TL [64] usando puerta lógicas CMOS convencionales para implementar las tres técnicas. La Figura 5.5 muestra los resultados para una simulación transitoria SPICE usando los modelos de los dispositivos de una tecnología AMS de 90nm, un voltaje de polarización V_{DD} de 0.5V y una temperatura de 100 ^{o}C. El ruido interno se genera mediante la opción de ruido transitorio, opción interna del simulador, la cual genera ruido intrínseco en cada transistor y en cada fuente de alimentación. El ruido generado por el simulador es amplificado en un orden de 100 veces. El voltaje nominal V_{DD} para una tecnología de 90nm es de 1V, pero en este experimento se decremento a 0.5V para emular las condiciones ruidosas y de bajo voltaje de tecnologías futuras, donde el ruido térmico llegará a ser más relevante.

Usando una implementación basada en Verilog-A se generan las señales de entrada con ruido gaussiano blanco aditivo (Aditive White Gaussian

Noise AWGN), con una media de $0V$ y una desviación estándar de 0.5 $Vrms$, utilizando semillas diferentes para cada fuente de ruido. De esta manera, se aplican fuentes de ruido no correladas a cada una de las entradas, mientras que las salidas de cada dispositivo bajo prueba tienen una puerta lógica NOT como carga.

Las formas de onda para una simulación transitoria son mostradas en la Figura 5.5, donde **xi** y **xic** son entradas comunes para todas las puertas lógicas bajo prueba, mientras que las salidas para CMOS estándar, MRF y TL son **xo_not**, **xo_mrf**, **xoc_mrf**, **xo_turtle** y **xoc_turtle**, respectivamente. Puede verse que TL tiene mejor robustez que las técnicas comparadas en ambientes donde los niveles de ruido tienen una magnitud similar a la magnitud V_{DD} de la señal en cuestión.

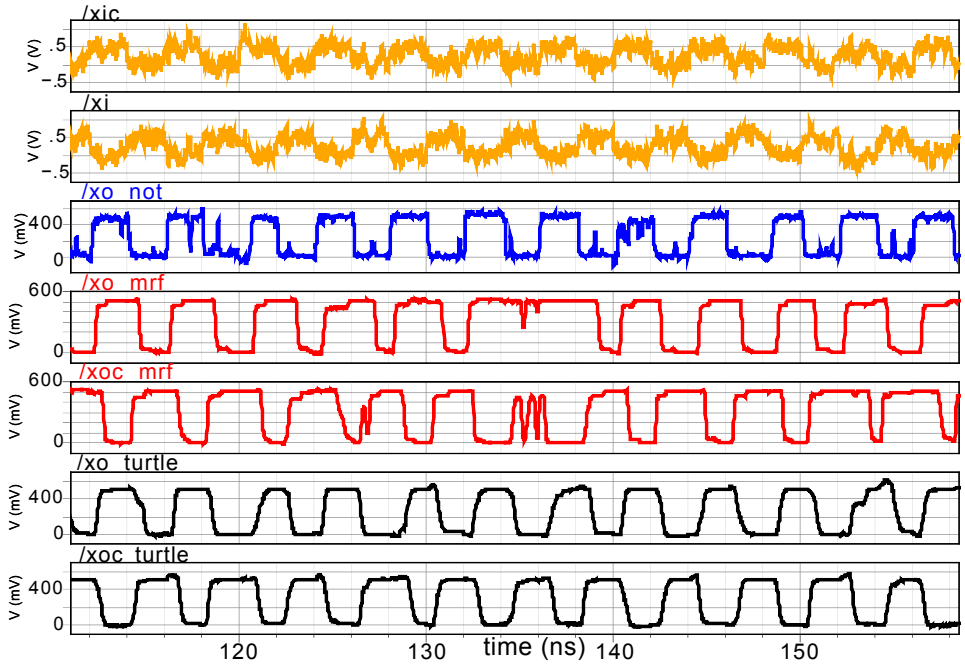


Figura 5.5: Resultados para una simulación dinámica para una puerta lógica NOT usando las técnicas CMOS estándar, MRF y TL.

La Tabla 5.3 muestra el coste en hardware así como el número de espurios que presenta cada una de las técnicas ante ruido para las puertas lógicas NOT, AND-NAND, OR-NOR y XOR-XNOR implementadas con tres técnicas diferentes, estándar CMOS, MRF y TL. Como parámetro de fiabilidad para medir y comparar la robustez de estas aproximaciones se usa el número de señales espurias (SS) presentes en el tiempo de simulación,

definidas en el trabajo de Moll en [65]. La generación de una señal espuria se realiza mediante la inyección de ruido blanco aditivo Gausiano (Additive White Gaussian Noise) AWGN con una media de 0V y una desviación estándar de 60m Vrms para los niveles lógicos "1" y "0". En orden a tener las mismas condiciones de simulación para las tres técnicas se utilizan todas las posibles combinaciones de entrada en una simulación transitoria, se suma el número de señales espurias que presenta cada técnica a la salida para una secuencia alternada de 1's y 0's con ruido AWGN. El experimento presentado en la Tabla 5.3 es toda señal no esperada que tiene un voltaje inesperado en un intervalo de tiempo $(t - \tau)$, donde τ es el tiempo mínimo necesario para que una señal de entrada se transfiera a su salida [65]. Una SS es contada cuando esta tiene una amplitud mayor que $(\frac{V_{DD}}{2} \pm 0.1V_{DD})$ y una anchura de tiempo de al menos el 10% del periodo de trabajo, que para este experimento es de 0.4ns.

En la Tabla 5.3 puede verse que para el caso de una puerta lógica NOT fueron contabilizados 8,298 espurios para la técnica CMOS estándar, 1,580 para el caso de MRF y 71 para la técnica TL. Por lo tanto, la puerta TL-NOT tiene mucho menor relación de error que estándar CMOS y MRF, con lo cual se demuestra que TL es una técnica mucho más robusta en ambientes extremadamente ruidosos.

Tabla 5.3: Comparativa de robustez para las técnicas CMOS estándar, Markov Random Field (MRF) y la Lógica Tortuga (TL) usando 5,000 1's y 0's alternados con un periodo T de 4ns para una puerta lógica NOT como medio de prueba.

Puerta lógica	Técnica	# Entradas	# Salidas	# Puertas	# SS para x_o & x_{oc}	
NOT-BUFFER	Static	1	1	1	8,298	N. A.
	MRF	2	2	8	1,580	2,850
	TL	2	2	10	71	73
AND-NAND	Static	2	1	2/1	5,778	10,864
	TL	4	2	18	596	939
OR-NOR	Static	2	1	2/1	4,061	13,351
	TL	4	2	18	930	1,163
XOR-XNOR	Static	2	1	5/5	12,313	14,545
	TL	4	2	18	1,683	1,867

5.5 Resultados de simulación para una implementación basada en transistores

En orden para validar la robustez de las puertas lógicas TL propuestas y cuya implementación es basada en transistores CMOS convencionales se implementan dos metodologías diferentes para las puertas lógicas NOT y NAND2. Los resultados de una simulación transitoria mostrados en las Figuras 5.6 y 5.7 corresponden a una simulación transitoria SPICE usando los modelos de dispositivo para una tecnología de 90nm, con un voltaje de alimentación V_{DD} de 0.15V a una temperatura de 100 ^{o}C. El ruido interno fue generado usando la opción de ruido-transitorio del simulador de circuitos SPECTRE©, el cual genera para cada paso en el tiempo ruido térmico, ruido de parpadeo (Flicker noise) en cada canal de cada transistor y ruido de disparo para cada fuente de alimentación [66, 67, 68, 69]. El ruido generado para cada transistor y para cada fuente de alimentación es amplificado 100 veces respecto al ruido intrínseca de la tecnología de 90nm, para emular el ambiente en futuras tecnologías que tendrán componentes muy ruidosos. Cabe señalar que el voltaje nominal V_{DD} para esta tecnología es de 1V, pero en este trabajo se decremento a un voltaje de 0.15V para simular las condiciones de trabajo de futuras tecnologías.

Las señales ruidosas de entradas son generadas usando una señal cuadrada a la cual se le adhiere una señal de ruido aditivo blanco Gausiano (Additive White Gaussian Noise) AWGN con una media de 0V y una desviación estándar de 60m Vrms, para ambos niveles lógicos (1-lógico y 0-lógico). Estas señales o fuentes de ruido de entrada son no correladas y se aplican a cada una de las entradas, utilizando una semilla diferente para cada fuente de entrada ruidosa, mientras que las salidas del dispositivo bajo prueba tienen idénticos elementos como carga.

Los resultados obtenidos son presentados en las figuras 5.6 y 5.7, dónde se observar que la lógica Tortuga tiene muy buena robustez en un ambiente dónde los niveles de ruido alcanzan una magnitud similar a V_{DD}, incluso cuando V_{DD} es menor que $|V_{TH}|$ para un modelo de operación de una tecnología de 90nm.

Como puede ser observado en la Tabla 5.4, en el caso de un diseño TL basado en transistores para el caso especifico para una puerta lógica NOT tiene un coste mayor en hardware que otras técnicas similares tal como la técnica “Differential Cascode Voltage Switch DCVS” pero la misma que la técnica basada en “Markov Random Field MRF”, sin embargo a medida que se incrementa la complejidad de las puertas lógicas TL la relación del coste

en hardware u "overhead" es cada vez menor respecto a las técnicas CMOS, DCVS y MRF reportadas por Nepal et. al en [63].

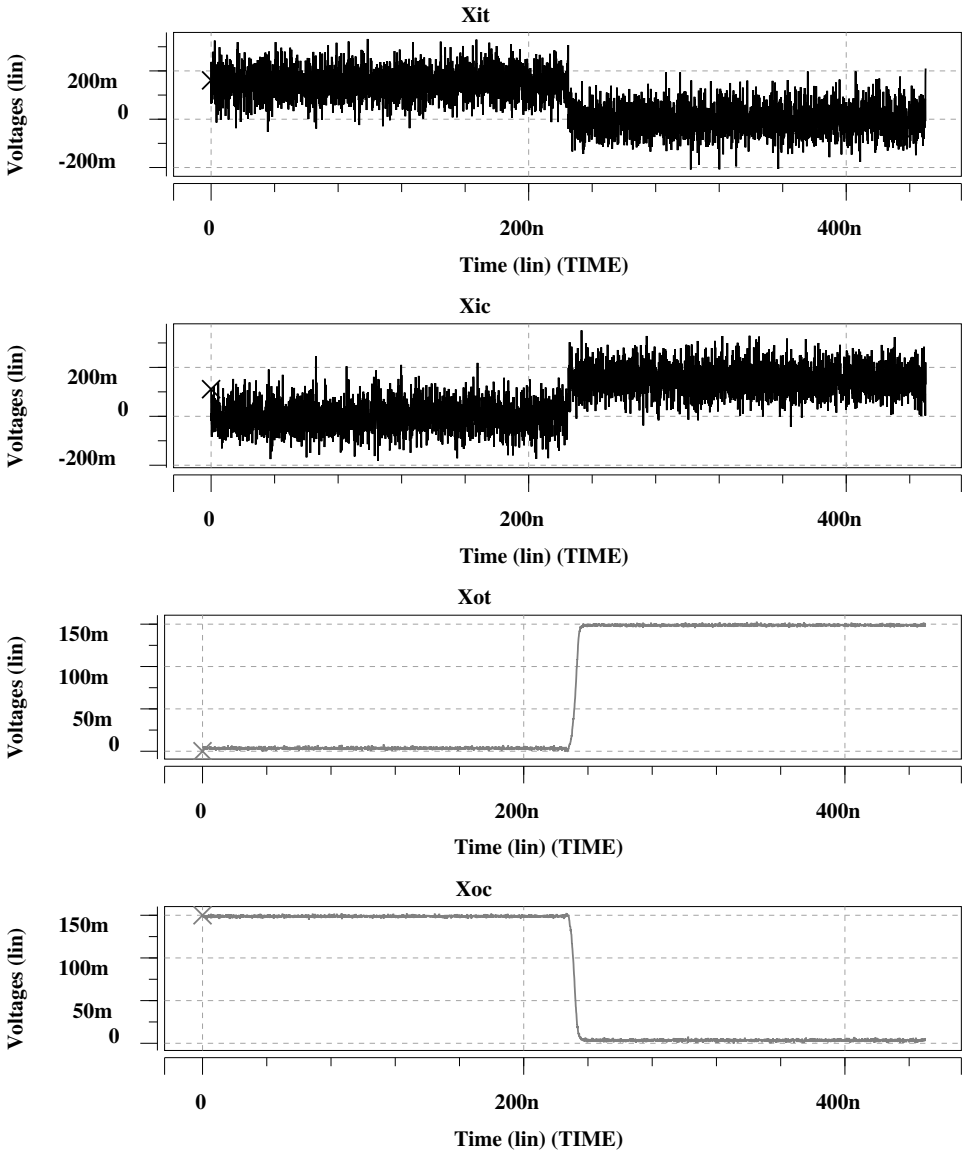


Figura 5.6: Resultados de una simulación transitoria para una puerta lógica NOT usando la Lógica Tortuga, con un voltaje de alimentación V_{DD} de 0.15V y con entradas ruidosas aplicando AWGN con una media de 0V y 60mV rms de desviación estándar. Donde, **Xit**-**Xix** son las señales de entrada, mientras que **Xot**-**Xoc** son las señales de salida.

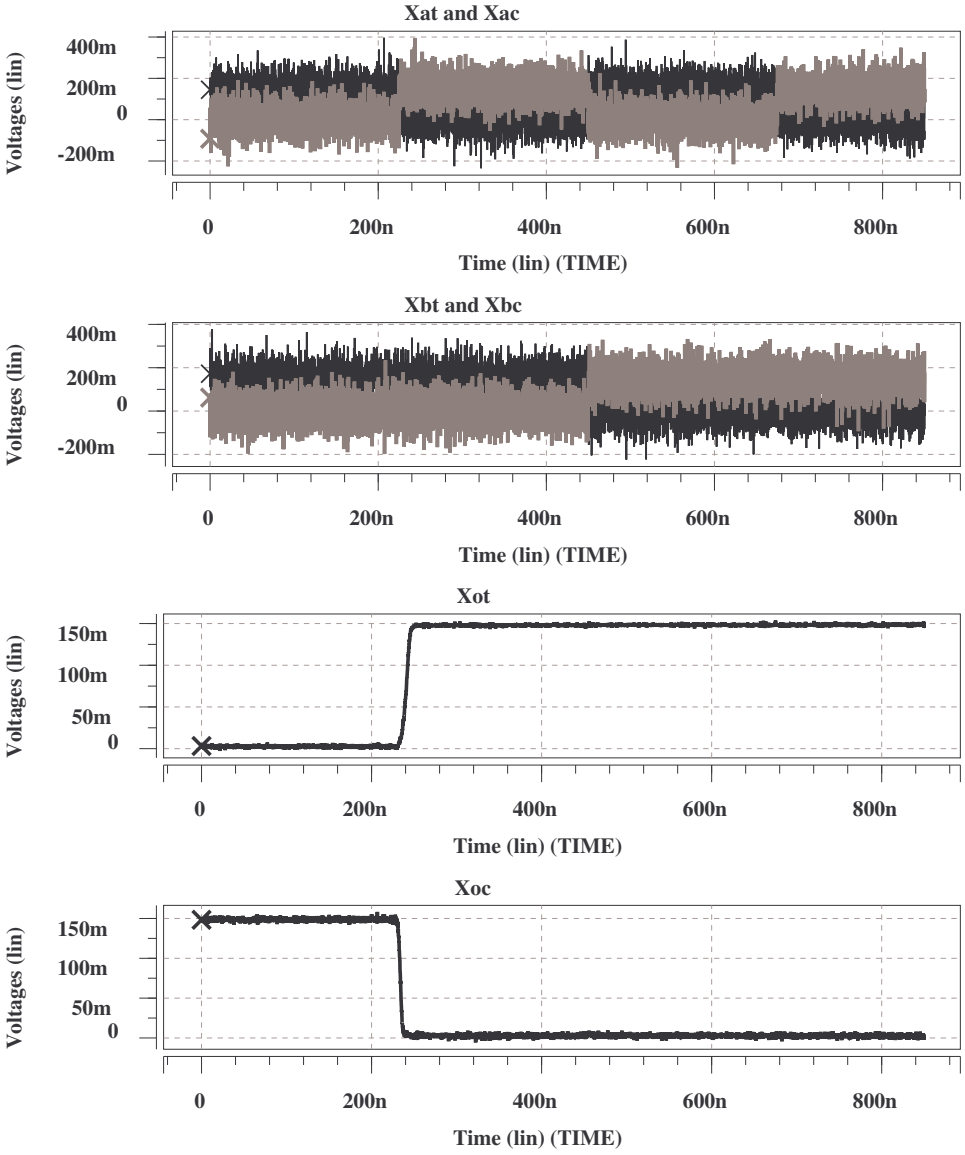


Figura 5.7: Resultados para una simulación transitoria de una puerta lógica NAND2 usando la Lógica Tortuga, para un voltaje de alimentación V_{DD} de 0.15V y con entradas ruidosas aplicando AWGN con una media de 0V y 60mV rms de desviación estándar. Se utilizan semillas diferentes para cada fuente. **Xat**-**Xac** y **Xbt**-**Xbc** son las señales de entrada, mientras que **Xot**-**Xoc** son las señales de salida.

Tabla 5.4: Comparativa del coste en hardware para los elementos lógicos de las técnicas CMOS, DCVS, MRF y TL.

Gate	Logic	Inputs	Outputs	Trans.	Overhead
NOT	Conv.	1	1	2	1.00
	DCVS	2	2	4	2.00
	MRF	2	2	20	10.00
	TL	2	2	20	10.00
NAND	Conv.	2	1	4	1.00
	DCVS	4	2	10	2.50
	MRF	4	2	60	15.00
	TL	4	2	32	8.00
NOR	Conv.	2	1	4	1.00
	DCVS	4	2	10	2.50
	MRF	4	2	58	14.50
	TL	4	2	32	8.00
XOR	Conv.	2	1	22	1.00
	DCVS	4	2	14	0.64
	MRF	4	2	60	2.72
	TL	4	2	48	2.00

5.6 Midiendo la inmunidad al ruido

Cualquier sistema está constituido por una serie de dispositivos interconectados de forma tal que son capaces de realizar unas funciones concretas. Estos bloques funcionales pueden estar constituidos por un único componente o por complejos subsistemas, dependiendo del tipo de sistema y de las interconexiones en el mismo. El estado de los componentes y la estructura del sistema determinan si un sistema está funcionando o no. En definitiva, el cuantificar la fiabilidad de un sistema requiere, generalmente, considerar la estructura del sistema y la fiabilidad de sus componentes.

En un sistema digital que tiene dos niveles lógicos “0” y “1”, la distancia de Kullback-Leibler (KLD) mide la diferencia entre la distribución de probabilidades de una señal real respecto a la distribución de probabilidades de una señal ideal, ecuación 5.1. De donde se establece que cuanto más pequeña es la medida KLD, mayor es la inmunidad a ruido del circuito que se esta midiendo. De donde se puede decir que un valor de KLD= 0 corresponde a una operación ideal y perfecta de los dispositivos. Por lo tanto, una posible

y apropiada medida de la inmunidad al ruido de un elemento lógico es la Distancia de Kullback-Leibler [70] (Kullback-Leibler Distance KLD) expresada por la ecuación 5.1, la cual determina la discrepancia entre la distribución de probabilidades de una salida ruidosa real (P_{real}) bajo prueba, respecto a la distribución de probabilidades de la misma salida ideal (P_{ideal}).

Realizando un muestreo de los voltajes de salida para este caso en especifico con periodo de 0.1ns, se mide y compara cuantitativamente la inmunidad a ruido de cualquier elemento lógico simple o complejo. Una comparación de KLD para una puerta lógica NOT, así como para una puerta lógica NAND2 implementadas mediante las técnicas estándar CMOS, DCVS, MRF y TL son mostradas en la Tabla 5.5. Dónde la lógica Tortuga de acuerdo a KLD es aproximadamente 3.6X, 13.4X y 20.9X veces mejor que MRF, DCVS y estándar CMOS, respectivamente. Esta medición da un valor LKD muy bajo para la Lógica Tortuga respecto a las otras tres técnicas, por lo tanto, una implementación TL tiene mejor inmunidad al ruido, pero con una penalización en hardware.

$$KLD(P_{ideal}, P_{real}) = \sum_{estados} P_{ideal} \cdot log_2 \left(\frac{P_{ideal}}{P_{real}} \right) \tag{5.1}$$

Tabla 5.5: Comparativa para las puertas lógicas NOT y NAND2 implementadas mediante las técnicas CMOS, DCVS, MRF y TL mediante de la distancia de Kullback-Leibler. Todas las posibles combinaciones lógicas para las entradas son generadas y se les adhiere una señal ruidosa generada por medio AWGN para una media de cero voltios y una desviación estándar de 60mV para un V_{DD} de 0.15V.

Lógica	NOT	NAND2
CMOS	3.4040	3.7947
DCVS	2.1832	3.6608
MRF	0.5878	0.4126
TL	0.1627	0.1881

La Lógica Tortuga (TL) es una metodología de diseño independiente de la tecnología. Esta puede ser usada para diferentes niveles de abstracción tal como transistor, puerta, RTL, alto-nivel, etc. Se puede utilizar en el diseño de diferentes niveles de elementos tal como puerta, circuitos combinacionales, elementos de memoria Latchs y/o Flip-Flops, sub-sistemas secuenciales e incluso en sistemas completos.

5.7 Lógica Tortuga: Circuitos combinacionales

Tomando como base los principios de diseño robusto de las puertas lógicas TL anteriormente expuestos, es posible implementar cualquier circuito combinacional. Para mostrar los principios de diseño del capítulo anterior, en la siguiente sección un bloque lógico, un ejemplo de implementación de un circuito combinacional es mostrado utilizando un sumador completo para dichos fines.

5.7.1 Sumador completo

Un sumador completo con entradas A, B y C_{in} y valores lógicos de salida S y C_{out}, está regido por las ecuaciones $S = A \oplus B \oplus C_{in}$ y $C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B))$ [71]. Remplazando cada puerta convencional por su puerta equivalente TL se obtiene el circuito de la Figura 5.8. El resultado es un circuito sumador completo (TL-FA), donde (a_t, a_c), (b_t, b_c), y (c_{it}, c_{ic}) son las entradas, y las salidas son (s_t, s_c) y (c_{ot}, c_{oc}). Cada puerto esta compuesto por sus señales verdadera ($_t$) y complementaria ($_c$).

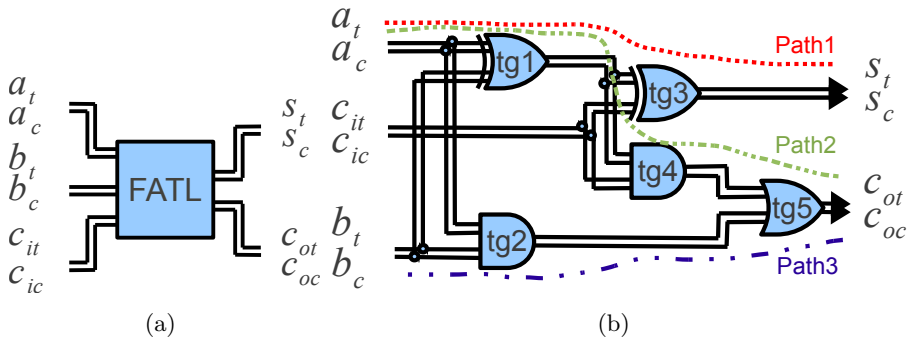


Figura 5.8: Sumador completo implementado con base a puertas lógicas TL. (a) Símbolo y (b) Vista esquemática.

5.7.2 Análisis de errores en un sumador completo TL

En esta sección se analiza el comportamiento funcional de un circuito combinacional TL cuando un error se inyecta en una de las rutas internas.

Suponemos que únicamente en la puerta *tg1* del sumador completo TL-FA Figura 5.8(b), ocurre una situación de incoherencia en una de sus en-

tradas. Debido a la naturaleza de las puertas lógicas TL (*tg1-tg5*), la puerta lógica *tg1* mantiene sus valores previos correctos, evitando la transferencia de información incorrecta. Sin embargo, como las puertas lógicas TL (*tg2-tg5*) se encuentran en condiciones de coherencia entre sus valores de entrada, ellas continúan con su operación lógica normal, causando que las demás puertas reciban y generen valores de salida coherentes, pero con información incoherente.

Para evitar la transferencia de datos incorrectos, que en este caso específico son valores coherentes es necesario un sistema de banderas (flags) que indiquen cuando las puertas TL están procesando valores correctos o no, se presenta un mecanismo de solución en la siguiente sección.

5.7.3 Error en una sola ruta en un circuito con multi-rutas

En cada puerta individual TL una señal de bandera complementaria es incluida la cual determina cuando las entradas de la puerta TL son coherentes y cuando no. El proceso de banderas complementarias son independientes de la operación de la puerta TL y son implementadas usando una puerta convencional XOR para la parte verdadera y con una XNOR para la parte complementaria, las cuales verifican la consistencia de cada par de entradas complementarias en cada puerta TL.

Una posible implementación del sistema de señales de bandera para una puerta lógica TL-NOT, es el circuito que se presenta en la Figura 5.9. Dónde, fg_t y fg_c son las banderas complementarias que validan los valores de salida respecto a los valores de entrada. Si y solo si, los valores lógicos de las banderas son fg_t=1 y fg_c=0, entonces los valores lógicos de salida de la puerta lógica son válidos; en cualquier otro caso los valores lógicos de salida no son válidos.

Para implementar circuitos combinacionales con el sistema de validación de banderas antes mencionado, se utiliza como medio de prueba un sumador completo que se presenta en la Figura 5.8 y se trabajó en la sección 5.7.1. Al cual se le aplica el sistema de banderas a cada una de las puertas lógicas TL se muestran en la Figura 5.10 y diferentes niveles de banderas tanto a nivel puerta como combinacional. El nivel de banderas para puerta lógica necesita un circuito que valide y determine cuando el dato de salida de un circuito combinacional es válido o no.

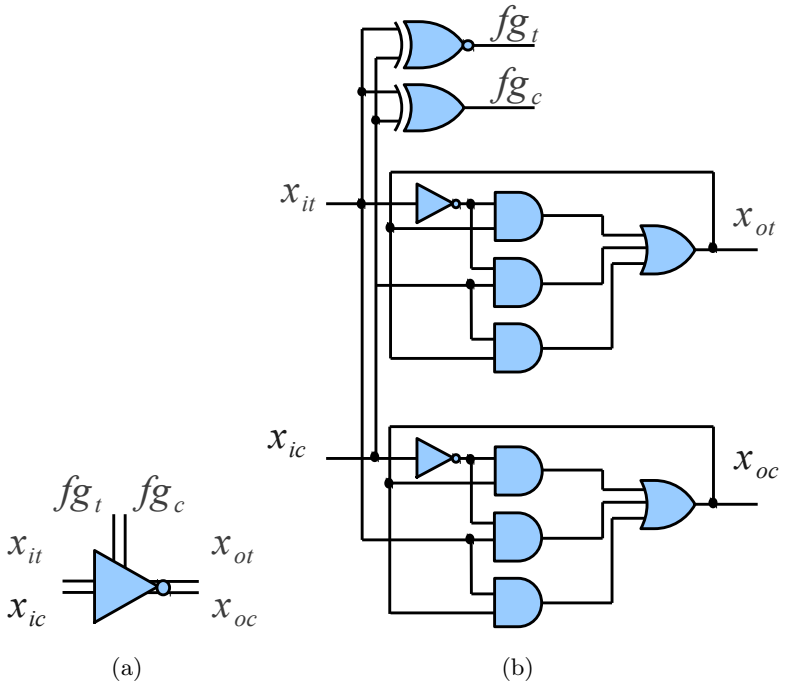


Figura 5.9: Puerta lógica TL-NOT con banderas que validan los datos. (a) Simbolo y (b) vista esquematica.

Para el caso especifico del sumador completo TL-FA, una posible implementación se muestra en la Figura 5.10(c). La cual recolecta todas las señales de bandera que son generadas por las puertas lógicas tg1, tg2, tg3, tg4, y tg5, validando que sean correctas $fg_t = 1$-$fg_c = 0$, los valores de las banderas lógicas toman los valores de $fl_t = 1$ y $fl_c = 0$. En caso de que almenos una de las banderas complementarias presente una incoherencia o presenten valores lógicos de que ha ocurrido algún error en alguna de las puertas lógicas tg1, tg2, tg3, tg4 o tg5, los valores de las banderas de salida son $fl_t = 0$ y $fl_c = 1$. De esta forma indicando que a nivel lógico combinacional ha ocurrido un error y por lo tanto, el valor de salida del sumador completo no es válido y no debe ser considerado. Para que se de la condición anterior, la condición que debe cumplirse para el sistema de banderas de las puertas lógicas, así como de los circuitos combinacionales TL, es que las banderas deben ser más rápidas que las señales de procesamiento de las puertas lógicas y que las señales de procesamiento de los circuitos combinacionales, para que de esta forma se genere la correcta señalización de error con miras a implementar circuitos secuenciales. El sistema de banderas que valida el correcto procesamiento de las puertas lógicas y que en conjunto implementan un elemento combinacional se presenta en Figura 5.10 inciso c), indicando por medio de las señales fl_t and fl_c, si los valores lógicos de

salida del elemento combinacional son válidas o no. Para el caso específico del sumador completo, estas señales son las que indican si los valores de suma y acarreo son correctas o no.

Con un sistema de banderas en ambos niveles de diseño lógico, a nivel de puertas lógicas (fg_t y fg_c) y a nivel de circuitos combinacionales (fl_t y fl_c), la lógica TL tiene un sistema jerárquico de banderas que habilita o no sus salidas.

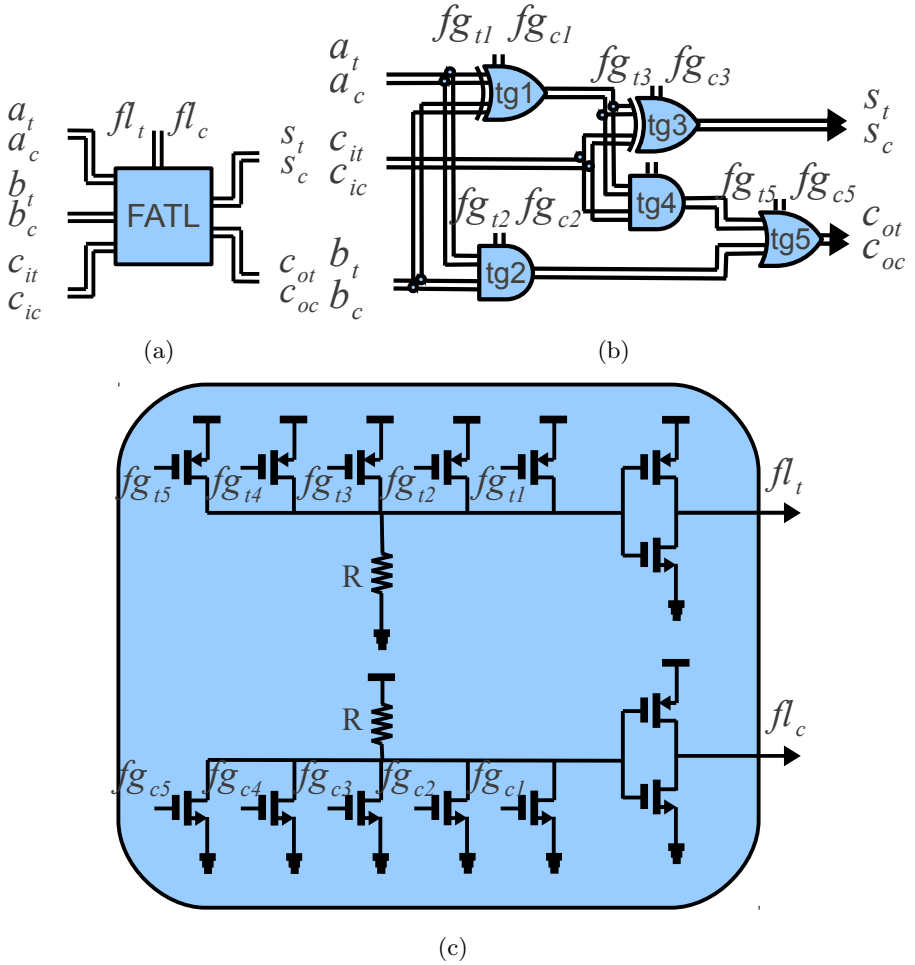


Figura 5.10: Sumador completo (TL-FA) usando puertas lógicas TL para su implementación así como el sistema de banderas que habilitan la valides de los datos. (a) Símbolo, (b) vista esquemática y (c) implementación propuesta para el sistema de banderas.

5.8 Conclusiones

Los resultados presentados en esta sección demuestran que la Lógica Tortuga, propuesta en esta tesis a nivel puerta lógica, presenta una excelente tolerancia a señales espurias respecto a las metodologías convencional o estática y las basadas en Markov Random Field (MRF), como se presenta en la Tabla 4.3., donde se procesan 1,000 datos con señales espurias, 5,000 unos lógicos y 5,000 ceros lógicos. Con un voltaje de alimentación V_{DD}=0.15 V y una temperatura de 100^0C. De donde la Lógica Tortuga para un diseño de puerta lógica NOT y buffer es aproximadamente 30.6X y 116.8X veces más tolerante a señales espurias que las lógicas MRF y estática CMOS, respectivamente. Realizando un comparativa respecto a las puertas AND2, NAND2, OR2, NOR2, XOR2 y XNOR2, se obtiene que la Lógica Tortuga es 9.7X, 11.6X, 4.4X, 11.5X, 7.3X y 7.8X más tolerante a señales espurias que la Lógica estándar CMOS.

El ruido generado para cada transistor y para cada fuente de alimentación es amplificado 100 veces respecto al ruido intrínseco para una tecnología de 90nm, para emular el ambiente de tecnologías futuras que tendrán componentes muy ruidosos. Cabe señalar que el voltaje nominal para una tecnología de 90nm es de 1V, pero en esta sección se decremento a un voltaje de 0.15V para simular las condiciones de trabajo de futuras tecnologías.

La lógica Tortuga para una puerta lógica NOT necesita más hardware, la cual refleja un coste de 1X, 5X, 10X mayor que las lógicas basadas en MRF, DCVS y estándar CMOS. Para una puerta lógica para las mismas técnicas, es menor que MRF en 0.5X y mayor para DCVS y CMOS en 3.2X y 8X, respectivamente. De igual forma para una puerta lógica NOR en 0.5X, 3.2X y 8X y finalmente de 3.1X, 0.9X y de 2X, para una puerta lógica XOR.

Una posible forma de medir la inmunidad al ruido de un elemento lógico es mediante la distancia de Kullback-Leibler, la cual mide la discrepancia entre una distribución de probabilidades de una salida real respecto a una distribución de probabilidades ideal. En un sistema digital donde se tienen dos niveles lógicos, la distancia de Kullback-Leibler mide la distancia entre la distribución de probabilidades de la salida real respecto a la distribución de probabilidad de una señal ideal. Entre más pequeña es la medida de Kullback-Leibler es mayor la inmunidad al ruido de un elemento digital. De acuerdo a los resultados presentados en la Tabla 4.5 de las medidas de Kullback-Leibler para un diseño estándar CMOS, uno basado en Differential Cascode Voltaje Swith (DCVS), uno basado en Markov Random Field (MRF) y uno en la Lógica Tortuga, la última técnica es aproximadamente 3.6X, 13.4X y 20.9X veces mejor que MRF, DCVS y estándar CMOS, res-

pectivamente.

Del análisis de errores para un sumador completo utilizando la Lógica Tortuga se establece que un sistema de banderas es necesario que indique cuando una dato de salida lógico es válido o no.

Capítulo 6

Lógica secuencial

Hasta ahora, se han analizado puertas lógicas y circuitos combinacionales, es decir, circuitos donde las salidas dependen única y exclusivamente de los valores de entradas, y no de la historia pasada del sistema, como ocurre en los sistemas secuenciales o de memoria temporal. Un sistema secuencial es un sistema digital que introduce dependencia temporal, donde al menos una de sus salidas en cualquier instante de tiempo depende de almenos una entrada en dicho instante y de la historia pasada equivalente a una secuencia de entradas. Lo que implica una serie de características inherentes a estos sistemas.

- Tienen uno o más caminos de realimentación, es decir, una o más señales internas o de salida se vuelven a introducir como señales de entrada. Gracias a esta característica se garantiza la dependencia de la operación con la secuencia anterior.
- Existe una dependencia explícita del tiempo, que se produce en los lazos de re alimentación antes mencionados. En estos lazos es necesario distinguir entre las salidas y las entradas re alimentadas. Esta distinción se puede traducir en un retraso de ambas señales (en el caso más ideal), el cual puede producirse mediante dos elementos:
  - Elementos de retraso, ya sean explícitos o implícitos debido al retraso de la lógica combinacional. Este retraso es fijo e independiente de cualquier señal.
  - Elementos de memoria, que son dispositivos que almacena el valor de la entrada en un instante determinado por una señal externa (reloj) y lo mantiene hasta que dicha señal ordene el almacenamiento de un nuevo valor.

La diferencia de comportamiento entre ambos elementos radica en que la salida del elemento de retraso es una copia de la señal de entrada; mientras que el elemento de memoria copia determinados instantes de la entrada (determinados por una señal externa - reloj), y no la señal completa, el resto del tiempo la salida no cambia de valor.

En este caso, la salida del elemento de retraso es una copia de la señal de entrada retrasada un determinado tiempo; mientras que la salida del elemento de memoria copia los valores de la entrada cuando la señal de control tiene una transición de subida o de bajada, por lo que la copia no es exacta de toda la señal, sino que sólo copia los valores en tiempos de interés. Por lo tanto, el modelo clásico de un sistema secuencial consta de un bloque combinacional, que generará la función lógica que queramos realizar, y un grupo de elementos de memoria con una serie de señales realimentadas, como se muestra en la Figura 6.1.

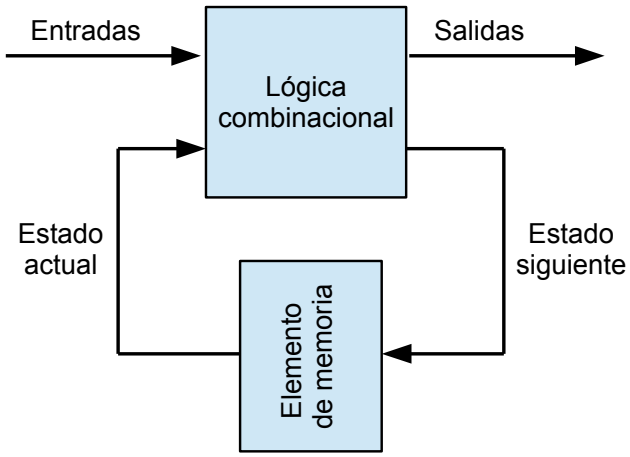


Figura 6.1: Modelo clásico de un sistema secuencial.

Se pueden distinguir tres tipos de señales: señales de entradas, señales de salida y señales de estado. Las señales de entrada y salida tienen el mismo significado que en los sistemas combinacionales. En cambio, las señales de estado son aquellas que mantienen la información de la historia pasada del sistema. Las señales de estado tienen dos versiones, según se consideren a la salida o a la entrada (estado actual; estado siguiente) del elemento de memoria:

- Si se consideran las señales de estado a la salida (estado actual) de los elementos de memoria, o lo que es lo mismo, a la entrada del

bloque combinacional, se denominan señales de *estado actual* ya que nos indica el estado en el que se encuentra el sistema para realizar una operación.

- Si se consideran las señales de estado a la entrada de los elementos de memoria, o lo que es lo mismo, a la salida del bloque combinacional, se denominan señales *estado siguiente* ya que nos indican el estado al que llegará después de que el bloque combinacional haya realizado la operación.

De acuerdo a las bases generales que gobiernan a los elementos de memoria que hasta este punto se han visto, las cuales son usadas para el diseño de los mismos elementos con base en las leyes que gobiernan a la Lógica Tortuga, para ello iniciamos desde la base del escenario actual de los dispositivos integrados, donde, las dimensiones de los dispositivos son escaladas agresivamente en cada generación tecnológica.

Para mantener la electrostática de los dispositivos y prevenir el rompimiento causado por los altos campos eléctricos, el voltaje de alimentación ha sido escalado a la par del escalamiento de las dimensiones de los transistores. Generando el efecto de reducir la capacitancia y el voltaje de alimentación en un nodo, de tal manera que la cantidad de carga almacenada en un nodo en particular es disminuido en cada generación. Esto ha generado un incremento en la fluctuaciones de voltaje y por lo tanto se incrementa el número de errores suaves en los elementos combinacionales y secuenciales [72].

Debido a la reducción de carga en cada nodo tecnológico, los elementos de memoria son cada vez más susceptibles a radiaciones cósmicas, provocando un Single Event Upset (SEU) es un evento no destructivo, sin embargo bajo ciertas circunstancias puede causar un corto circuito aparente de la línea de alimentación a referencia o tierra. Esta condición es referida como un Lathup (Single Event Latchup), el cual en la fabricación de dispositivos que no cuenten con contra medidas en su diseño pueden ser destruidos o dañados los dispositivos, como puede ser el caso de la ruptura de una puerta (Single Event Gate Rupture SEGR o Single Event Burnout SEB), todos ellos efectos de la radiación en dispositivos electrónicos llamados Single Event Effects (SEEs). Uno de los efectos de un SEE es denominado SET (Single Event Transient) el cual es una transitorio provocado por radiación solar o cósmica que puede ser generado y propagado a través de la cadena lógica de cualquier circuito, llevando un error a cualquier ruta lógica, provocando un error lógico el cual, puede llegar a propagarse hasta un elemento de memoria [73, 74].

Una radiación (SEU) es un cambio de estado causado por el golpe de una partícula ionizada simple (iones, electrones, fotones, ...) en una zona sensible

de un dispositivo, circuito o sistema electrónico, tal como elemento o circuito secuencial, memoria o microprocesador. El cambio de estado es resultado de una carga libre creada por la ionización en un nodo importante de un elemento lógico. Los parámetros del circuito y la energía de la partícula determinan si una partícula puede o no causar un un transitorio de tensión con suficiente energía para causar un error en la salida de un dispositivo.

Diferentes técnicas de diseño y modelado se han propuesto para ayudar a mitigar este problema. Se ha propuesto el uso de clonación de puerta y cambiar el tamaño de puerta como una alternativa menos costosa en comparación con la técnica redundancia triple modular, con el objetivo de mejorar la tolerancia a SET en lógica combinatoria con una tecnología CMOS [75, 76, 77]. Estas técnicas evalúan la mayoría de los nodos más vulnerables de un circuito basado en la probabilidad eléctrica de ocurrencia de un SET, en una ventana de vulnerabilidad o enmascaramiento en un circuito lógico o de latcheo. Estás mismas técnicas una vez que seleccionan los nodos de puerta más sensibles, se cambia el tamaño de esas puertas de tal manera que se incrementa la capacitancia del nodo para reducir la probabilidad de ocurrencia de un SET. El problema es que el aumento del nodo capacitancia no siempre puede dar lugar a una anchura reducida de un SET de tal manera que no genere un error en un circuito lógico o un latch.

Las fluctuaciones en la lógica digital de circuitos integrados pueden afectar a los estados lógicos de sus nodos, resultando en algunas ocasiones en el cambio de estado de los valores lógicos [72]. De esta manera, se puede crear un efecto transitorio que puede corromper la lógica dentro de un circuito. Si un SET (Single Event Transient) es capturado por un determinado elemento lógico, entonces se puede generar un error suave. Entiéndase error suave, como un valor lógico erróneo que es transitorio y no permanente, con la suficiente energía para generar un cambio lógico en un determinado nodo, corrompiendo la lógica en un circuito digital. Como se muestra en la Figura 6.2, los latches en un Flip-Flop son vulnerables a SETs durante el tiempo de latch (T_{setup}) y un periodo T_{tf} (ciclo opaco) durante el cual un fault transitorio puede ocurrir. Estos periodos son llamados como la ventana de vulnerabilidad (Window of Vulnerability-WOV)[16]. Donde, T_{nvd} es el periodo cuando el latch es transparente: T_{prop} es el tiempo donde un fault afecta un nodo dentro del latcheo y se propaga a la salida.

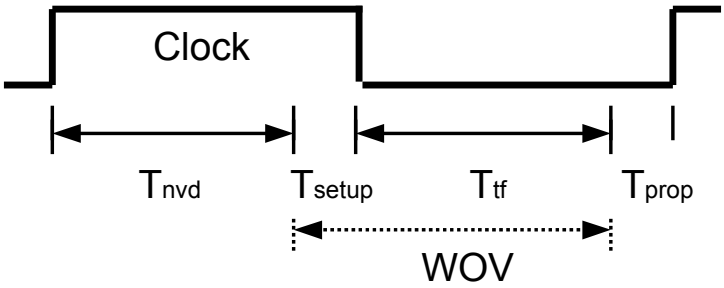


Figura 6.2: Ventana de vulnerabilidad (WOV) [16].

Errores suaves o fluctuaciones de tensión no deseadas pueden afectar de igual manera tanto a memorias, elementos secuenciales como elementos lógicos. Convencionalmente, las técnicas basadas en el principio de paridad y códigos correctores de errores (ECC) son usadas para detectar y corregir algunos errores suaves en la transmisión de datos [78]. Sin embargo, elementos secuenciales como Latches y Flip-Flops son difíciles de proteger [79, 80]. Técnicas para incrementar la flexibilidad o tolerancia a errores suaves en latches y Flip-Flops han sido propuestas por Hazucha y Zhaou en [79, 81]. Estas técnicas incrementan la capacitancia de determinados nodos, utilizando redundancia para mejorar la tolerancia a errores suaves. Sin embargo, estas técnicas tienen un coste en términos de retardo, área, consumo de energía y complejidad.

Con el objetivo de conocer la relevancia que tienen los elementos secuenciales en materia de tolerancia a errores suaves, Mitra en [17] precisa que los elementos secuenciales contribuyen con el 49% de toda la relación de errores suaves en un circuito. Para ello es importante considerar que los latches en un Flip-Flop son sometidos a ciclos continuos de estados de transparencia y opaco, equivalente a que el estado de transparencia un Latch tipo D transfiera un valor lógico de su entrada a su salida, y para el caso del estado opaco permanece con su valor anterior sin importar el valor actual en su entrada.

6.1 Latches y Flip-Flops tolerantes a SEUs

El Latch es el elemento básico por el cual se puede realizar la construcción de un Flip-Flop, por ejemplo es posible crear un D-Flip Flop utilizando dos D-Latches conectados en cascada. En esta sección se presentan trabajos previos que tienen como objetivo proporcionar tolerancia a errores suaves para latches y Flip-Flops. Así mismo, se presenta un bloque básico para la

construcción de sistemas secuenciales para lo cual se realiza la descripción de un *basic scan Flip-Flop* y un *scan hold Flip-Flop*.

Dentro de los elementos secuenciales existen esquemas para almacenar réplicas de datos correctos durante la operación de los mismos. Este método los hace más robustos a radiaciones externas y consecuentemente más tolerantes a sus efectos sobre memorias.

Diseños que combaten estas corrupciones han sido trabajados en [16] por Omana, en [82] por Arima, en [83] por Rockett, en [84] por Whitaker y en [85] por Liu, los cuales utilizan hardware adicional para almacenar una copia de los datos para ser utilizados o retroalimentados en los puntos de procesamiento de interés. Estos diseños proporcionan tolerancia solo a SETs durante el periodo T_{tf} de una WOV y tienen un alto coste en área.

Los latches diseñados en [86] por Calin y en [87] por Naseer tienen mayor retardo, mayor complejidad y un coste reflejado en un incremento del número de transistores así como en un incremento en el tiempo de retardo. Un latch con una topología resistente a un evento simple (single event resistant topology - SERT) es presentado en [88] por Gambles, el cual almacena valores de los datos verdaderos así como complementarios en cuatro nodos.

Posibles soluciones limitadas por el coste en el número de Latches utilizan redundancia y muestreo en intervalos de tiempo reflejados en un coste en retardo (delay) para el caso combinacional y un coste de perdida de ciclos de reloj, como es mostrado por Nicolaidis y Mavis en [89, 90].

Por otro lado, Nicolaidid en [89] sugiere el uso de código de palabras de estado conservando los elementos o bloques (code word state preserving - CWSP) mediante lo cual se bloquea la propagación de errores. Estos diseños tienen un enorme coste en términos de área y una reducción en las prestaciones de los circuitos.

Una forma común de tolerancia a errores suaves es el uso del principio de redundancia triple modular (TMR) [24]. La National Aeronautics and Space Administration (NASA) ha patentado el diseño de registros tolerantes a errores suaves [91] basado en este principio. En [92, 93] Krishnamohan presenta un esquema de votación mayoritaria síncrona que utiliza redundancia espacial sobre nodos cargados para lograr una determinada tolerancia. Los diseños de Krishnamohan son vulnerables a radiaciones sobre sus nodos de salida que dinamicamente almacenan una carga.

Otras técnicas hacen uso de escaneo de hardware redundante, para bloquear e identificar la ocurrencia de un SET en un Flip-Flop, conocidas como

"Error blocking scan Flip-Flop"(EBSFF), "error trapping scan Flip-Flop" (ETSFF), y "error blocking scan hold Flip-Flop"(EBSHFF), presentadas por Mitra en [17], las cuales proporcionan un esquema de bajo coste en transistores para la detección y bloqueo de errores suaves, por medio del uso de buffers en las salidas.

Una modificación de la técnica "scan design Flip-Flop" (LSSD) atiende la corrección de errores LSSD en Flip-Flops la cual proporciona tolerancia a SETs del contenido de los latches en el caso de un error. Si ocurre un SET en cualquier otro estado entonces por medio de votación mayoritaria se refresca el dato o contenido de los Latches. Sin embargo, los diseños de Mitra, Goel y Drake presentados en [17], [18] y [19], respectivamente, son vulnerables a SETs durante el T_{setup} de su Latch maestro y cuya condición de error se ilustrada en la Figura 6.3.

Un Flip-Flop basado en redundancia de tiempo es propuesto en [94], el cual incorpora un circuito de decisión que ayuda a la selección de la salida a través de un multiplexor, presentando un gran coste en términos de prestaciones y área.

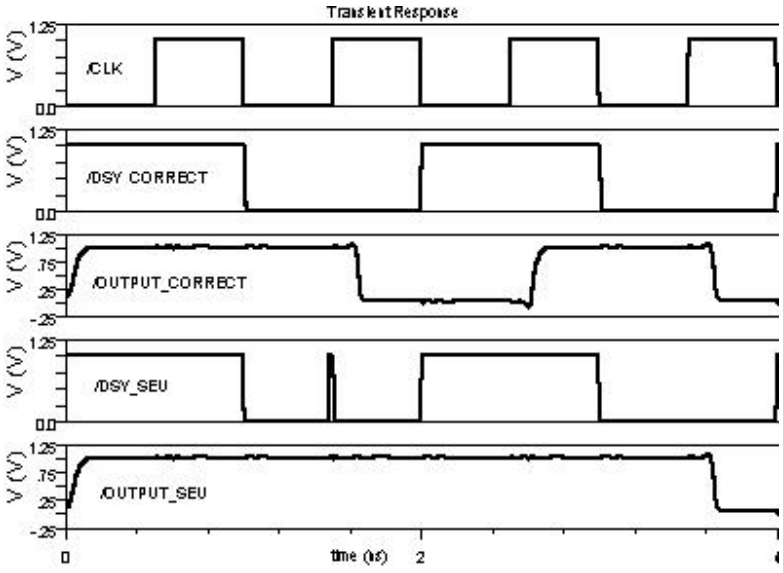


Figura 6.3: Condición de error para los diseños de Mitra [17], Goel [18] y Drake [19].

Roystein en [20] adhiere un latch redundante en la estructura existente

de un Flip-Flop y el dato es registrado o almacenado en múltiples latches. El contenido de estos múltiples latches son entregados a un votador mayoritario, y si el contenido de alguno de estos latches es corrupto por un error suave, este es filtrado a la salida a través del mismo circuito de votación mayoritaria. El diseño de Roystein emula una redundancia triple modular para un Flip-Flop con la re utilización de un circuito de escaneo, ver Figura 6.4. Así, el diseño de Roystein utiliza un incremento de dos latches comparado a los cuatro en un esquema TMR. Roystein propone para el caso de incremento de área, reducirla por medio de la eliminación de un latch haciendo uso de un scan hold Flip-Flop, entregando así una estructura Flip-Flop enfocada a:

- Operación en modo funcional con tolerancia a errores sobre la ventana de vulnerabilidad WOV.
- Escaneo básico basado en pruebas para fallas estructurales.
- Pruebas de retardo para escaneo estructural.

Donde un posible diagrama a bloques para un Flip-Flop mediante la técnica TMR tradicional es ilustrada en la Figura 6.4. El Flip-Flop es replicado tres veces para procesar en paralelo por triplicado el dato de entrada D y cuyas salidas son entregadas a un votador mayoritario para obtener un valor mayoritario de salida.

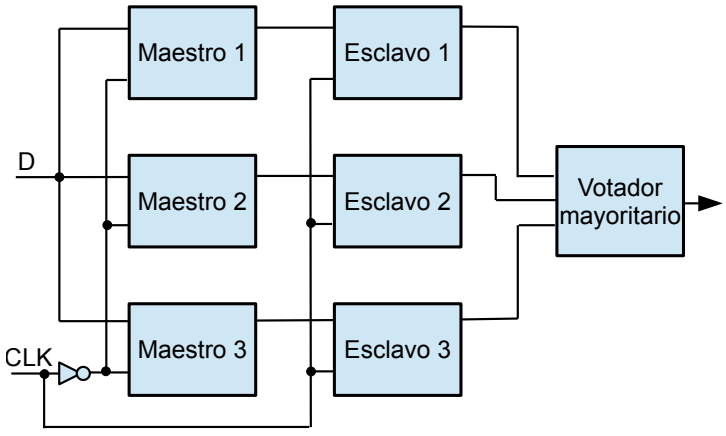


Figura 6.4: Esquema general para un Flip-Flop utilizando la técnica Redundante Triple Modular (TMR) [20].

En orden para diseñar máquinas de estados lógicos es necesario contar

con los elementos básicos Latch y Flip-Flop, por lo tanto, en las siguientes secciones se presentan el diseño de estos elementos con base a los principios de la Lógica Tortuga.

6.2 Latch tipo D

Con base en los principios de la Lógica Tortuga que hace uso de la redundancia de puertos PR-1, como primer paso se replican con su señal complementaria los puertos de entrada y salida en un Latch tipo D. Resultando en que los datos y el reloj tengan una redundancia complementaria. Las ecuaciones que gobiernan el comportamiento lógico para un Latch tipo D con transparencia para un reloj con valor de cero lógico ($CLK = 0$) y asegurando el dato ("Latching"), con un valor de reloj uno lógico ($CLK = 1$) basado en los principios lógicos de la Lógica Tortuga son:

- Si ($CLK_t = 0$ & $CLK_c = 1$ & $(D_t \neq D_c)$), entonces
  - $Ql_t = D_t$,
  - $Ql_c = D_c$.
- Para cualquier otro caso:
  - $Ql_t = Ql_t prev$,
  - $Ql_c = Ql_c prev$.

donde, D_t y D_c son las señales redundantes de entradas, CLK_t y CLK_c son las señales redundantes de relojes, Qlx son las salidas redundantes. Los sub-índices $_t$ y $_c$ corresponden a las señales redundantes verdadera y complementaria, respectivamente.

A partir de las ecuaciones previas que gobiernan el comportamiento de un latch tipo D con base en los principios de lógica Tortuga, se generaron los valores lógicos que se presentan en la Tabla de verdad 6.1. Dónde la primera columna representa los estados en decimal, segunda y tercer columna son los datos redundantes complementarios de entrada, cuarta y quinta columna son el reloj, sexta y septima el dato de salida previo y finalmente las dos últimas columnas son el dato actual.

Considerando los datos de entrada-salida y relojes redundantes, los estados lógicos en decimal 20, 21, 22, 23, 36, 37, 38 y 39 corresponden a datos

de entrada y señales de reloj coherentes (renglones sombreados en la Tabla de verdad 6.1), que corresponde a estados lógicos coherentes válidos, en los cuales el Latch tipo D Tortuga transfiere los datos de entrada a la salida de acuerdo a las ecuaciones que lo gobiernan, obtenidas a partir de la Tabla de verdad 6.1.

Los demás casos en decimal corresponden a estados donde ocurre una discrepancia o error ya sea en los datos de entrada o las señales de reloj, generando que el Latch tipo D Tortuga, mantenga sus datos de salida previos como se establece en la Tabla de verdad 6.1. El símbolo que representa un Latch tipo D Tortuga se presenta en la Figura 6.5(a). Como resultado de la síntesis de la Tabla de verdad 6.1 se obtiene el circuito combinacional de la Figura 6.5(b), el cual tiene un dato de entrada redundante, un reloj redundante y un dato de salida redundante, donde el último se retroalimenta a la entrada de forma lógica, permitiendo verificar la coherencia tanto del dato de entrada, salida y del reloj, y por medio de la lógica combinacional es posible tomar la decisión más adecuada de acuerdo a las ecuaciones que gobiernan a un Latch tipo D Tortuga. Una posible forma de implementar un Latch tipo D Tortuga mediante una implementación a nivel de transistores, se presenta en la Figura 6.5(c), donde se logra la propiedad de mantener valores previos de salida cuando ocurre una incoherencia, ya sea en el reloj o en el dato de entrada por medio de la retroalimentación de un inversor lógico de la señal de salida redundante a un nodo de interés previo, como se puede observar en la Figura 6.5(c).

Tabla 6.1: Tabla de verdad para un Latch tipo D activo en nivel bajo, basado en los principios lógicos de la Lógica Tortuga (TL-D Latch). Los renglones sombreados, correspondes a valores de entrada y reloj coherentes.

Estado	D_t	D_c	CLK_t	CLK_c	Ql_tprev	Ql_cprev	Ql_t	Ql_c
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0	1
2	0	0	0	0	1	0	1	0
3	0	0	0	0	1	1	1	1
4	0	0	0	1	0	0	0	0
5	0	0	0	1	0	1	0	1
6	0	0	0	1	1	0	1	0
7	0	0	0	1	1	1	1	1
8	0	0	1	0	0	0	0	0
9	0	0	1	0	0	1	0	1
10	0	0	1	0	1	0	1	0

Continúa en la siguiente página

Tabla 6.1 – continuación de la página anterior

Estado	D_t	D_c	CLK_t	CLK_c	Ql_tprev	Ql_cprev	Ql_t	Ql_c
11	0	0	1	0	1	1	1	1
12	0	0	1	1	0	0	0	0
13	0	0	1	1	0	1	0	1
14	0	0	1	1	1	0	1	0
15	0	0	1	1	1	1	1	1
16	0	1	0	0	0	0	0	0
17	0	1	0	0	0	1	0	1
18	0	1	0	0	1	0	1	0
19	0	1	0	0	1	1	1	1
20	0	1	0	1	0	0	0	1
21	0	1	0	1	0	1	0	1
22	0	1	0	1	1	0	0	1
23	0	1	0	1	1	1	0	1
24	0	1	1	0	0	0	0	0
25	0	1	1	0	0	1	0	1
26	0	1	1	0	1	0	1	0
27	0	1	1	0	1	1	1	1
28	0	1	1	1	0	0	0	0
29	0	1	1	1	0	1	0	1
30	0	1	1	1	1	0	1	0
31	0	1	1	1	1	1	1	1
32	1	0	0	0	0	0	0	0
33	1	0	0	0	0	1	0	1
34	1	0	0	0	1	0	1	0
35	1	0	0	0	1	1	1	1
36	1	0	0	1	0	0	1	0
37	1	0	0	1	0	1	1	0
38	1	0	0	1	1	0	1	0
39	1	0	0	1	1	1	1	0
40	1	0	1	0	0	0	0	0
41	1	0	1	0	0	1	0	1
42	1	0	1	0	1	0	1	0
43	1	0	1	0	1	1	1	1
44	1	0	1	1	0	0	0	0
45	1	0	1	1	0	1	0	1
46	1	0	1	1	1	0	1	0
47	1	0	1	1	1	1	1	1
48	1	1	0	0	0	0	0	0
49	1	1	0	0	0	1	0	1

Continúa en la siguiente página

Tabla 6.1 – continuación de la página anterior

Estado	D_t	D_c	CLK_t	CLK_c	Ql_tprev	Ql_cprev	Ql_t	Ql_c
50	1	1	0	0	1	0	1	0
51	1	1	0	0	1	1	1	1
52	1	1	0	1	0	0	0	0
53	1	1	0	1	0	1	0	1
54	1	1	0	1	1	0	1	0
55	1	1	0	1	1	1	1	1
56	1	1	1	0	0	0	0	0
57	1	1	1	0	0	1	0	1
58	1	1	1	0	1	0	1	0
59	1	1	1	0	1	1	1	1
60	1	1	1	1	0	0	0	0
61	1	1	1	1	0	1	0	1
62	1	1	1	1	1	0	1	0
63	1	1	1	1	1	1	1	1

Con el objetivo de mantener los principios que gobiernan a la Lógica Tortuga es necesario contar con un elemento de validación del dato de salida, se propone el uso de puertas lógicas XOR convencionales, las cuales en conjunto multiplican y suman lógicamente las señales verdadera y complementaria, para obtener banderas redundantes complementarias que validan la coherencia del reloj y del dato de entrada, determinando cuando el dato de salida redundante es válido o no. Las ecuaciones que describen el comportamiento lógico de un sistema de banderas para el Latch tipo D de la Figura 6.5(d) son:

- Si $(CLK_t \neq CLK_c)$ & $(D_t \neq D_c)$), entonces
  - $fdl_t = 0$,
  - $fdl_c = 1$,
- Para cualquier otro caso:
  - $fdl_t = 1$,
  - $fdl_c = 0$.

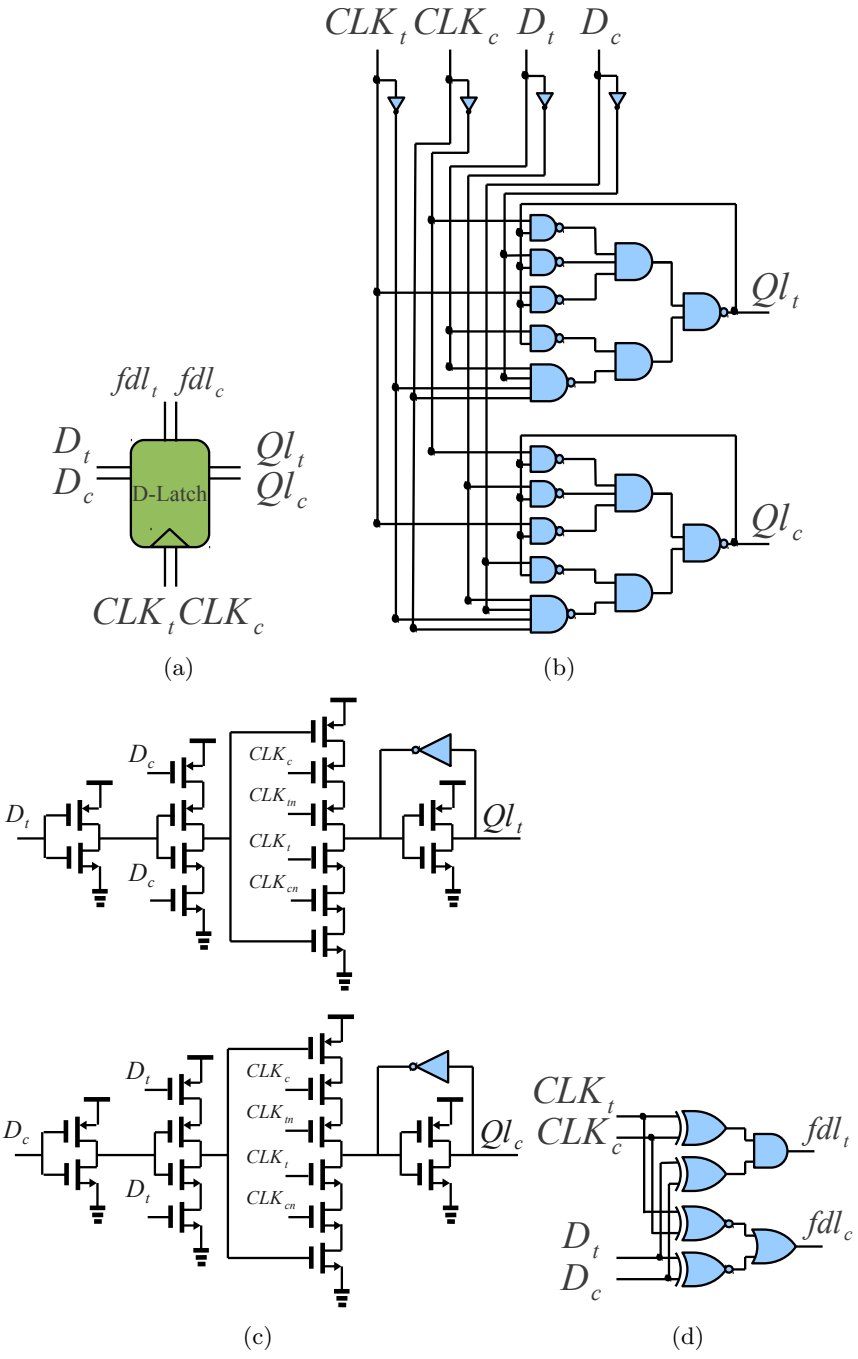


Figura 6.5: Latch tipo D Tortuga. (a) Símbolo, (b) posible diseño a nivel puerta, (c) posible diseño a nivel transistor y (d) implementación del sistema de banderas de validación de la coherencia en el dato de entrada y reloj redundante. Este sistema de validación se utiliza tanto para una implementación a nivel puerta lógica como a transistores.

Este sistema de banderas trabaja de manera independiente al sistema de procesamiento de los datos, el cual valida la coherencia de los datos de los elementos secuenciales, cabe mencionar que este sistema de banderas es necesario para cada elemento secuencial así como para cada elemento combinacional, respetando un orden jerárquico de procesamiento.

6.3 Flip Flop tipo D

La diferencia entre un Flip-Flop D y un Latch D es que el Flip-Flop copia la entrada D a la salida Q en el flanco del pulso de reloj, mientras que el Latch lo hace por nivel. El Flip-Flop tipo D es un elemento de memoria que puede almacenar información en forma de un 1 o 0 lógico.

Una posible implementación de un Flip-Flop tipo D Tortuga es por medio de dos Latch tipo D Tortuga conectados en cascada, como se muestra en la Figura 6.6(b), por lo tanto el Flip-Flop resultante hereda los principios y especificaciones de la lógica Tortuga, esto significa que el Flip-Flop resultante tiene el doble de puertos de entrada, salida y control o reloj. Con base al diseño de un Flip-Flop basado en dos Latch conectados en cascada, el reloj del segundo Latch debe ser invertido, y considerando que la naturaleza redundante complementaria de la lógica Tortuga, la inversión lógica del reloj se logra invirtiendo físicamente las conexiones de las señales CLK_t y CLK_c, eliminando la puerta lógica NOT.

Las señales fdf_t y fdf_c son las banderas que indican cuando los datos de salida del Flip-flop tipo D Tortuga son válidos o no y una posible implementación es presentada en la Figura 6.6(d).

La ecuaciones que describen el comportamiento de un Flip-Flop D activado por flanco de subida (TL-D Flip-Flop) son:

- Si $((CLK_t \uparrow \parallel CLK_c \downarrow)$ & $(D_t \neq D_c))$, entonces
  - $Qf_t = D_t$,
  - $Qf_c = D_c$.
- Para cualquier otro caso:
  - $Qf_t = Qf_tprev$,
  - $Qf_c = Qf_cprev$.

dónde, D_t y D_c corresponden a las señales de entrada redundantes, CLK_t y CLK_c son las señales de relojes redundantes y Qfx son las señales de salida. No importa si $CLK_t \uparrow$ ocurre antes, después o durante $CLK_c \downarrow$, los datos de entrada no son transferidos a las salidas hasta la última ocurrencia de cualquiera de los relojes redundantes. Resultado de la síntesis de las ecuaciones anteriores se obtienen los circuitos que se presentan en la Figura 6.6.

Con el objetivo de mantener los principios que gobiernan la Lógica Tortuga, se genera un sistema de banderas que validan la coherencia de los datos y se presenta en la Figura 6.6(d) las cuales son:

- Si $(CLK_t \neq CLK_c)$ & $(D_t \neq D_c)$), entonces
  - $fdf_t = 0$,
  - $fdf_c = 1$,
- Para cualquir otro caso:
  - $fdf_t = 1$,
  - $fdf_c = 0$.

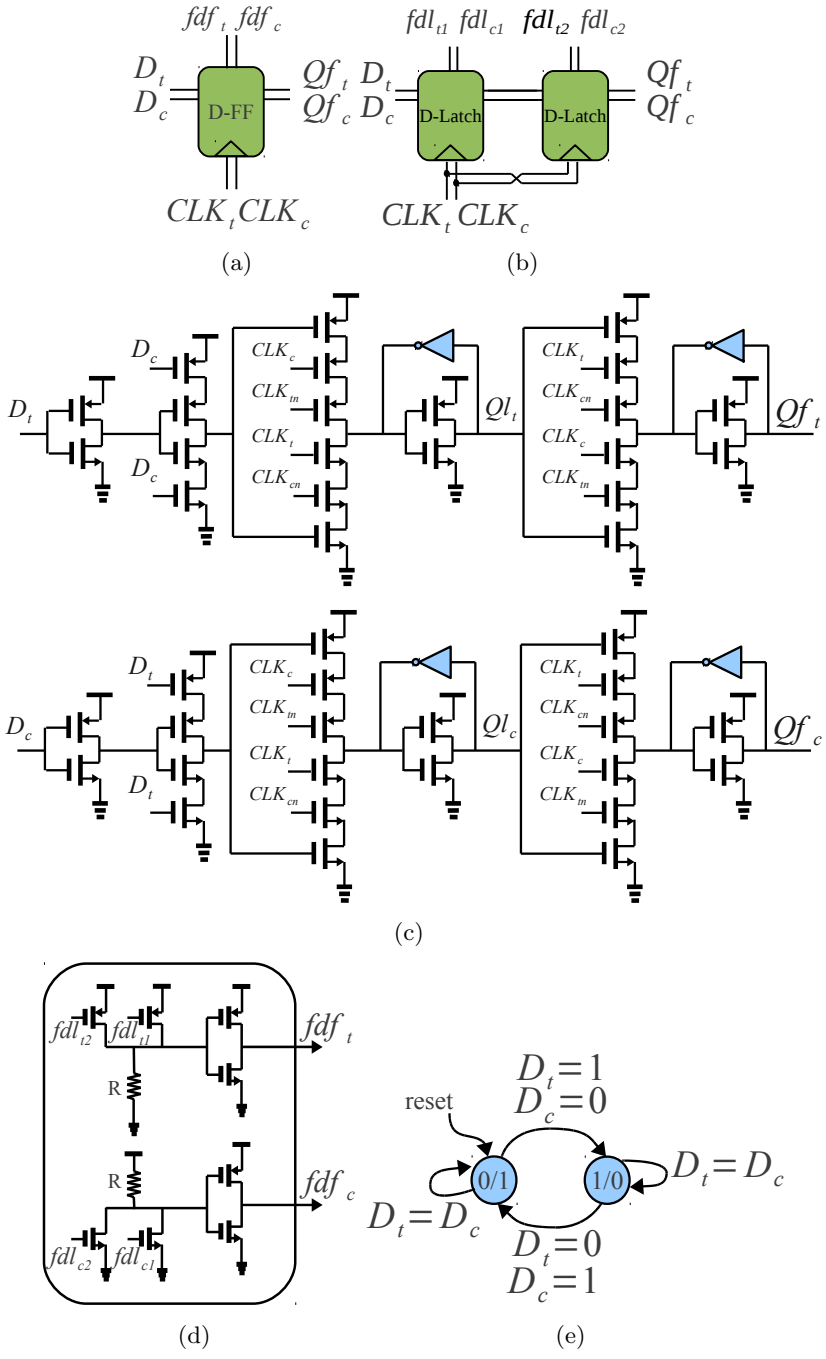


Figura 6.6: TL-D Flip-Flop. (a) símbolo, (b) vista esquemática usando dos TL-D Latches conectados en cascada, (c) implementación a nivel transistor, (d) implementación del sistema de banderas y (e) máquina de estados finitos (Finite State Machine FSM). Cabe señalar que los incisos (b) y (d) o (c) y (d) trabajan en conjunto para la implementación del inciso (a), para una implementación a basada en puertas o basada en transistores, respectivamente.

La Tabla 6.2 presenta el coste en hardware correspondiente para cada circuito implementado mediante diferente técnica o alternativa mediante una implementación a nivel puerta así como mediante celdas estándares. Comparado el coste en área para un Latch tipo D implementado mediante la técnica Tortuga respecto a CMOS convencional, TMR, 1^{st} CTMR and MRF, es respectivamente 5.8X, 0.84X, 0.27X y 0.56X veces mayor. La misma comparación para un Flip-Flop D es respectivamente 7.0X, 1.05X, 0.34X y 0.56X veces mayor. Cabe resaltar, que el coste en área de los elementos secuenciales tratados en esta sección, fueron obtenidos usando celdas estándares para una implementación a nivel puerta lógica, que corresponde al peor escenario en términos de coste de área. En términos del coste en hardware la Lógica Tortuga presenta mejores características en comparación con estrategias redundantes tal como TMR, CTMR y MRF.

Tabla 6.2: Comparación del coste de área de los elementos secuenciales implementados mediante la Lógica Tortuga, respecto a las técnicas de implementación CMOS convencional, TMR, 1^{st} CTMR and MRF. La primer línea de la Tabla, representa el número de transistores para cada elementos secuencial, mientras que la segunda línea expresa el coste en área expresado en μm^2.

	CMOS	TMR	1^{st} CTMR	MRF	TL
D-Latch	8T	74T	248T	172T	32T
	168	1162	3641	1744	979
D-FF	17T	128T	410T	334T	64T
	275	1866	5752	2386	1958

6.4 Resultados de simulación para elementos secuenciales TL

En orden de validar el mejoramiento de la robustez de los elementos secuenciales TL propuestos Figuras 6.5 and 6.6, se implementaron los elementos lógicos D-Latch y D-Flip Flop mediante dos metodologías: estándar CMOS y Lógica Tortuga. Los resultados obtenidos de una simulación SPICE son presentados en la Figura 6.7, usando los modelos de los dispositivos de una tecnología de 90nm con un V_{DD} de 0.5V a una temperatura de 127 ^{o}C. El ruido intrínseco e interno es generado usando la opción de transient-noise del simulador, el cual genera ruido térmico y ruido Flicker en cada canal de cada transistor así como ruido de disparo para cada fuente de alimentación para cada instante de tiempo o paso de simulación. El ruido generado para

cada transistor y para cada fuente de alimentación es amplificado 100 veces respecto al ruido intrínseco y propio de la tecnología de 90nm, con el objetivo de simular las condiciones de trabajo de tecnologías futuras las cuales tendrán componentes muy ruidosos. Cabe resaltar que el voltaje nominal para la tecnología de 90nm V_{DD} es de 1V, pero en este experimento fue disminuida a 0.5V para que de igual forma se tengan condiciones de trabajo de tecnologías futuras dónde la relación señal a ruido (SNR) será más relevante.

Las señales de entrada ruidosas son generadas usando módulos AWGN implementados usando Verilog-A, con una media de 0V y una desviación estándar de 0.5 Vrms para ambos valores lógicos "1" y "0". De forma similar son generadas las señales redundantes complementarias de los relojes CLK_t y CLK_c. Se usan semillas diferentes para cada módulo generador de señales ruidosas con el objetivo de generar fuentes ruidosas sin relación, mientras que las salidas del dispositivo bajo prueba operan libremente y con elementos secuenciales idénticos como carga.

Las formas de onda de una simulación transitoria son mostradas en la Figura 6.7 dónde CLK_t y CLK_c (CLKt y CLKc) son las señales redundantes complementarias de los relojes, D_t y D_c (dt y dc) son las señales de los datos de entrada para todos los dispositivos. Q_DL_cmos Ql_t, Qf_t, son las señales de salida redundantes complementarias para el TL-D Latch y TL-D Flip Flop, respectivamente. Las simulaciones se realizaron para un V_{DD} de 500mV.

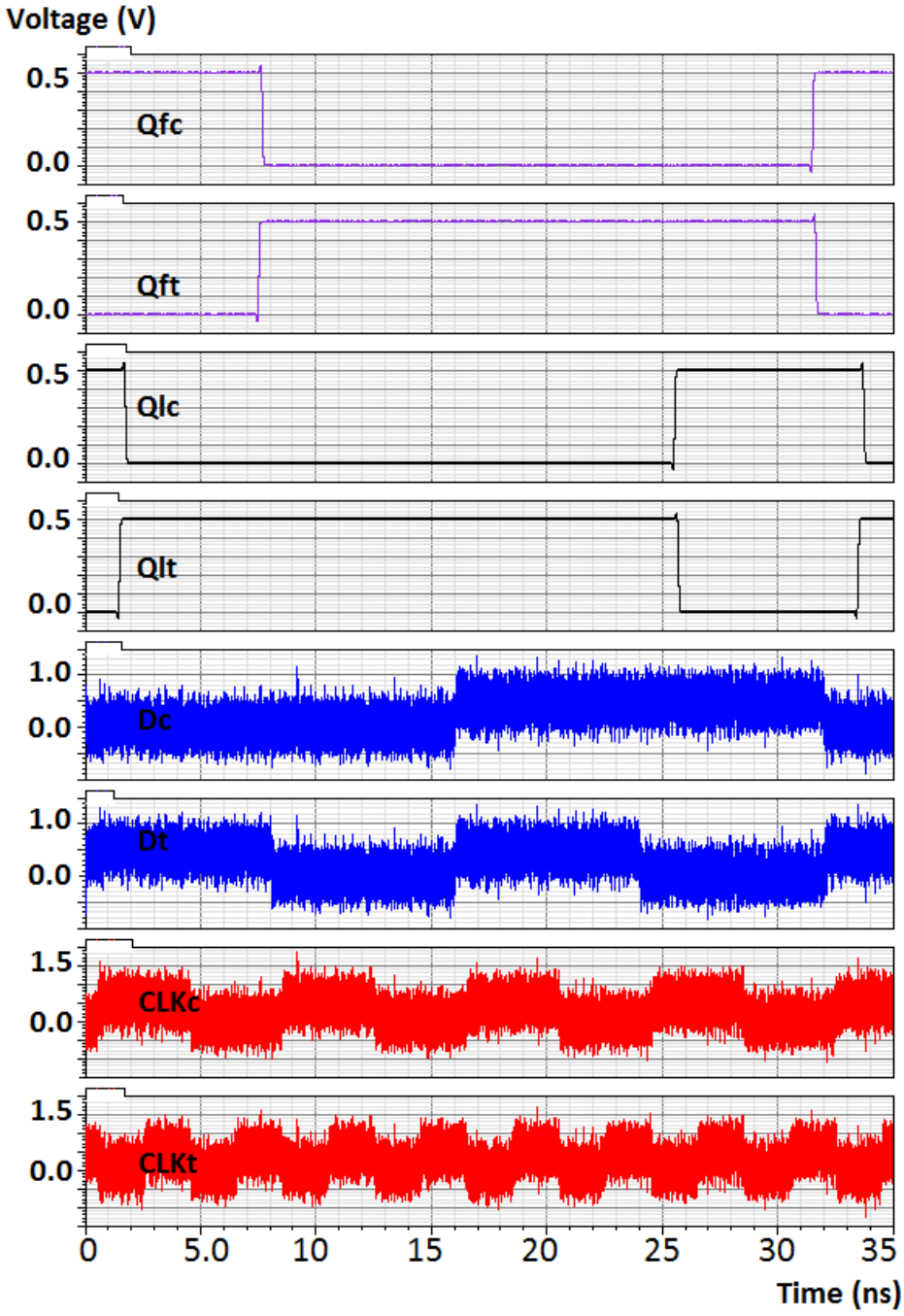


Figura 6.7: Comparativa de los resultados transitorios para un D-Latch y un D-Flip Flop implementados con CMOS y la Lógica Tortuga, todos ellos con entradas ruidosas mediante una fuente ruidosa AWGN con media de cero voltios y una desviación estándar de 500mV rms, sumadas a las entradas D and CLK tanto a la línea verdadera como a la complementaria.

6.4.1 Midiendo la inmunidad al ruido

Utilizando la Distancia de Kullback-Leibler (KLD) como parámetro de medida y gobernada por la ecuación 5.1, la cual ha sido presentada en la sección 4.3. Este parámetro determina la discrepancia entre la distribución de probabilidades de la salida ruidosa real del circuito bajo prueba, respecto a la distribución de probabilidades ideal de la misma salida, Tabla 6.3. Cabe resaltar que entre menor es el parámetro de la Distancia de Kullback-Leiber el dispositivo bajo prueba, presenta mejor tolerancia al ruido, siendo el caso ideal cero.

Tabla 6.3: Comparación de la tolerancia de error de un Flip-Flop tipo D implementado mediante las técnicas Triple Modular Redundancy (TMR), 1^{st}-orden de Cascade Triple Modular Redundancy (CTMR), Markov Random Field (MRF) reinforcer y Lógica Tortuga (TL).

	CMOS	TMR	1^{st} CTMR	MRF	TL
D-Latch	1.401	0.5913	0.4745	0.2481	0.1070
D-FF	1.2204	0.5699	0.4309	0.2190	0.0930

La Tabla 6.3 muestra los factores de KLD para un Latch tipo D y un Flip-Flip D para cinco casos de implementación. Los elementos secuenciales basados en la Lógica Tortuga contra las técnicas de acuerdo a las técnicas Triple Modular Redundancy (TMR) [24], 1^{st}-order Cascaded Triple Modular Redundancy (1^{st} CTMR) [25] and Markov Random Field (MRF) [27], presentan mucho mejor tolerancia a ruido de acuerdo a la distancia de Kullback-Leibler.

6.5 Conclusiones

Los elementos secuenciales basados en la Lógica Tortuga heredan las bondades de la lógica Tortuga en términos de inmunidad al ruido, así como penalizaciones en términos de área.

Se ha introducido un técnica de diseño redundante para bloques secuenciales que presentan mejor robustez a aplicaciones con altos niveles de ruido y bajo voltaje, tal como TMR, CTMR o MRF. Se ha obtenido experiencia con la implementación de elementos secuenciales sobre técnicas VLSI obteniendo mayor robustez.

Con el objetivo de emular el comportamiento de futuras tecnologías se realizó la experimentación con diferentes fuentes de ruido, tal como ruido térmico, ruido flicker y ruido de disparo. Estos ruidos o factores son críticos para aplicaciones en el ámbito de las comunicaciones, navegación y control. Realizando una comparación de la tolerancia al ruido para un Flip-Flop D, por medio de la Distancia de Kullback-Leibler, que determina la discrepancia entre la distribución de probabilidades de la salida ruidosa respecto a la distribución de probabilidades de la salida ideal, se obtuvo que la lógica Tortuga es 2.3X, 4.4X, 5.5X y 13.1X, mejor que las técnicas MRF reinforcer, 1st-order CTMR, TMR y estándar CMOS, respectivamente. En el mismo sentido una implementación de una Flip-Flop D presenta 2.3X, 4.6X, 6.12X y 13.1X veces mejor robustez al ruido que las técnicas MRF reinforcer, 1st-order CTMR, TMR y estándar CMOS, respectivamente.

Se ha mostrado que los elementos secuenciales basados en la Lógica Tortuga exhiben mucho mejor inmunidad al ruido de acuerdo a la medida de Kullback-Leibler. Estos circuitos presentan un coste adicional en área respecto a una implementación CMOS estándar, pero mucho menor respecto a las técnicas basadas en estrategias probabilísticas, así como a una arquitectura Von Neumann. Por lo tanto, elementos basados en la Lógica Tortuga mejoran la fiabilidad de los sistemas en escenarios ruidosos a un coste razonable, llegando a ser una atractiva alternativa.

Capítulo 7

Un procesador Tortuga

En esta sección, se considera el caso general donde un procesador es construido con base a elementos básicos, combinacionales y secuenciales diseñados y construidos con base en los principios de la Lógica Tortuga. Por lo tanto, el nuevo elemento "Procesador Tortuga" hereda las propiedades intrínsecas de la Lógica Tortuga que ya han sido discutidas ampliamente en capítulos previos. Los cuales de manera general son detener el procesamiento de datos, reloj y señales de control, ya sea por la incoherencia en sus entradas externas o internas, de cualquier modulo o nivel de procesamiento, no así de manera global y/o total de sus elementos así como entre procesadores. De lo anterior se determina que es necesario un protocolo de comunicación específico "Handshake" que asegure los principios lógicos de comunicación entre procesadores síncronos con los mismos principios de la Lógica Tortuga.

Para el escenario de trabajo de este capítulo el procesador Tortuga se encuentra operando entre dos procesadores Tortuga de características similares, Procesador 1 y 3 como se presenta en la Figura 7.1. Resaltando que hay una diferenciación entre los procesadores 1 y 3 del procesador 2 solo con la finalidad de no confundir la funcionalidad de un procesador basado en los principios que rigen a la Lógica Tortuga.

7.1 "Hanshake" para la Lógica Tortuga

Describir protocolo de comunicación básico y como se extrapola a tortuga

Como medio de prueba para el protocolo de comunicación "Handshake"

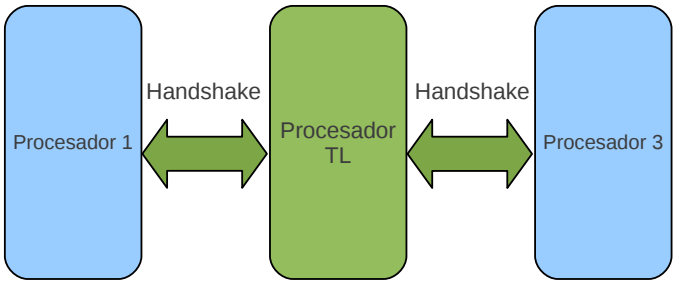


Figura 7.1: Procesador basado en la Lógica Tortuga que opera entre dos procesadores de características similares.

es utilizado un procesador con cuatro estados "pipeline" el cual es presentado en la Figura 7.2. A este procesador le es adaptado el protocolo de comunicación de Sparso [95], el cual utiliza un esquema de control tradicional de "acknowledge" *Ack* y "request" *Req* para la comunicación entre máquinas síncronas vecinas.

Se substituyen las señales de control *Ack* y *Req* por señales redundantes y complementarias de acuerdo a la Lógica Tortuga, las cuales verifican el adecuado control de comunicación y correcto procesamiento-cómputo de datos, en los cuatro estados "pipeline" del Procesador Tortuga. Cuando ocurre una discrepancia ya sea en las señales de "Handshake" *Ack* y *Req*, elementos redundantes de la lógica combinacional Tortuga, banderas redundantes "Flag" y/o dispositivo de control con señales redundantes, se generan señales de alerta tanto en el estado "pipeline" previo como en el siguiente. Generando las señales redundantes de alerta para los procesadores 1 y 3, indicando que el dato actual dentro de la cadena de procesamiento puede ser erróneo y por lo tanto debe desecharse para el procesador 3 y repetirse por el procesador 1. De esta manera el procesador 1 genera el dato previo correcto y es enviado al procesador Tortuga en el periodo de reloj siguiente. En el mismo sentido, el procesador 3 desecha el dato actual proveniente del Procesador Tortuga ya que corresponde a un dato incorrecto.

Resultado de la adaptación del protocolo de comunicación "Handshake" de Sparso en un procesador Tortuga con cuatro estados "pipeline" se obtienen los módulos de Lógica Combinacional Tortuga; de Control; de Registro de almacenamiento temporal $RegH$; y las banderas de indicación de datos correctos provenientes de la lógica combinacional Tortuga "Flag", ver Figura 7.2. Que en resumen se describen como:

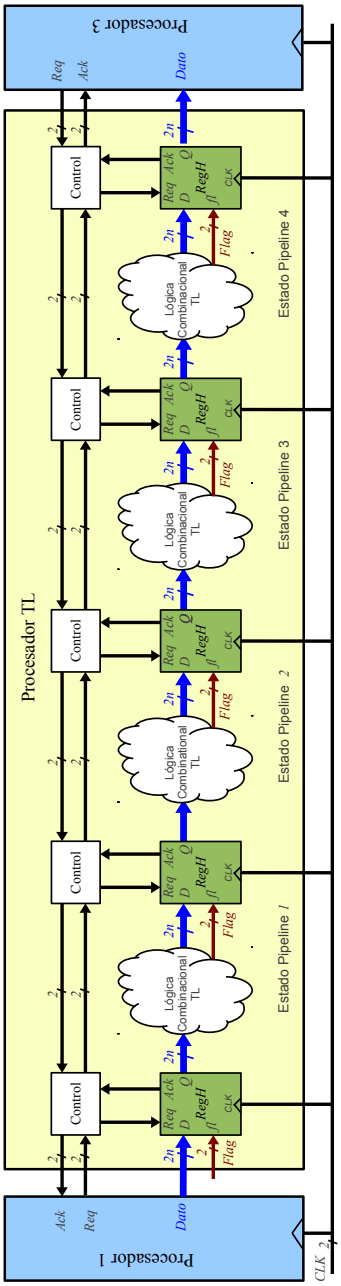


Figura 7.2: Procesador general implementado mediante la Lógica Tortuga con 4 estados pipeline y protocolo de comunicación “Handshake”.

7.1.1 Lógica Combinacional TL

Lógica Combinacional TL, es el conjunto de elementos Combinacionales TL que realizan una función booleana específica, por ejemplo, un sumador com-

pleto. Esta Lógica Combinacional TL corresponde a la que fue tratada en la sección 5.7, la cual tiene la propiedad de detener el cómputo de los datos por medio de señales redundantes complementarias y validadas por las banderas complementarias de "Flag".

7.1.2 Control

Una posible forma de implementar el módulo de "Control" del "Handshake" es implementado por medio de puertas lógicas NAND2, que validan las señales que provienen ya sea de un estado "pipeline" previo o siguiente, o de los procesadores 1 o 3, ver Figura 7.2. Las puertas NAND2 procesan y validan las señales redundantes *Ack* y *Req* del módulo de control, son puertas lógicas que permiten reducir el retardo de validación respecto al retardo del procesamiento de la información de un Procesador TL. Esta reducción del tiempo de retardo permite que incoherencias en las señales de bandera sean detectadas antes que sea procesado síncronamente un dato. Por lo tanto, se coadyuva a la pronta detección de una incoherencia en los datos procesados síncronamente y en las señales *Ack* y *Req* del módulo de control. Como consecuencia se puede reducir el número de datos computados erróneamente de algún estado "pipeline" de un Procesador Tortuga.

7.1.3 Bandera de validación *Flag*

Son señales redundantes que validan los datos procesados por la lógica combinacional Tortuga. Es importante resaltar que el sistema interno de validación de banderas del procesador TL, opera de forma independiente de las señales redundantes de reloj. Con lo cual una vez que se produce alguna incoherencia en cualquier señal redundante de la Lógica Combinacional el sistema genera la bandera redundante que indica que ha ocurrido una incoherencia. Avisando al registro *Regh* del evento para que, por medio del protocolo de comunicación Handshake" avise a los registros vecinos, y estos a su vez, generen la señal correspondiente, tanto a los demás registros *Regh* como al procesador vecino.

7.1.4 Registro *RegH*

En orden para asegurar que el dato correcto sea entregado por el mecanismo de "Handshake" Tortuga a una siguiente etapa "pipeline" o en su caso al siguiente procesador, es necesario un registro de almacenamiento temporal

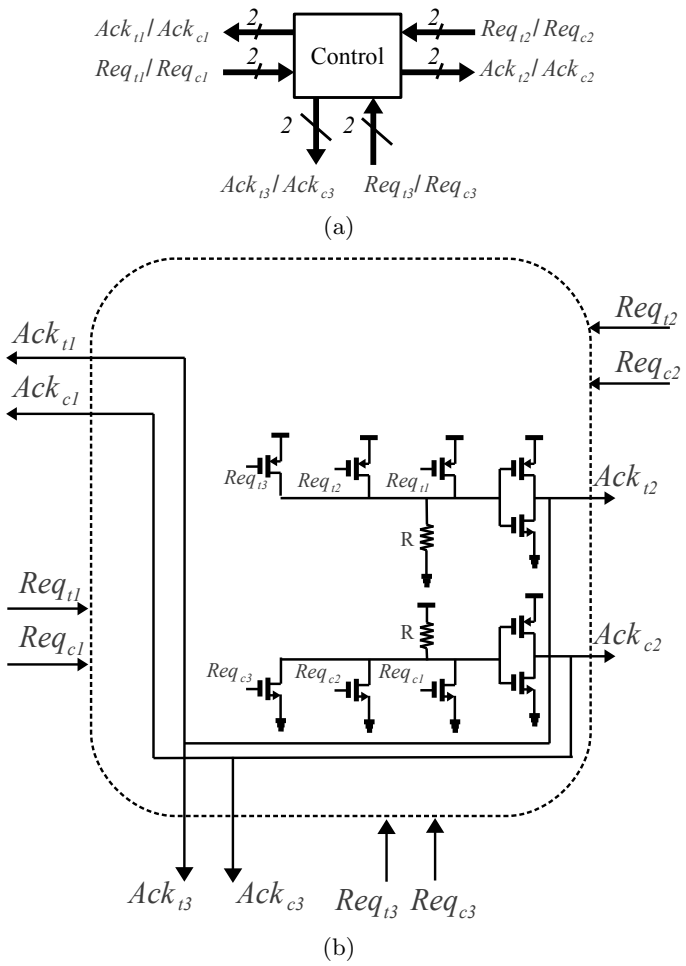


Figura 7.3: Módulo de Control del protocolo de comunicación "Handshake".

con funciones especiales. Este registro es llamado *RegH* el cual opera como registro de almacenamiento temporal (Flip-Flop tipo D) y al mismo tiempo toma en consideración el estatus-coherencia de las banderas de los estados anterior y siguiente, el estatus-coherencia de las señales redundantes *Req* y *Ack*, la coherencia de los datos de salida del bloque combinacional Tortuga y la coherencia de las banderas de validación de los datos del mismo bloque combinacional Tortuga. La descripción comportamental del registro *RegH* se estable como:

- Si $((CLK_t \uparrow \parallel CLK_c \downarrow)$ & $(CLK_t \neq CLK_c)$ & $fl_t = 1$ & $(fl_t \neq fl_c)$ & $Req_t = 1$ & $(Req_t \neq Req_c)$ & $(D_t \neq D_c))$, entonces
  - $Q_t = D_t$,
  - $Q_c = D_c$,

    * $Ack_t = 1$,
    * $Ack_c = 0$.

- Cualquier otro caso:
    * $Q_t = Q_t previo$,
    * $Q_c = Q_c previo$.
    * $Ack_t = 0$,
    * $Ack_c = 1$.

En dónde (D_t y D_c) son los datos redundantes en un procesador "pipeline" general, con una longitud de palabra de n bits redundantes para una posible implementación. (fl_t y fl_c) es la bandera redundante que indica si el dato de salida de un circuito Combinacional Tortuga fue procesado de forma correcta o no. Si ($fl_t = 1$ y $fl_c = 0$), entonces los datos son correctos. Si ($fl_t = 0$ y $fl_c = 1$) ha ocurrido una incoherencia entre los datos redundantes y por lo tanto, el sistema de comunicación "Handshake" Tortuga deberá operar hasta que el ruido que lo generó desaparezca, y de esta manera se corrija el dato. Finalmente, si ($fl_t = fl_c$), entonces existe una incoherencia en las banderas y de igual forma que para el caso de una incoherencia entre datos redundantes, el protocolo de comunicación "Handshake" Tortuga debe detener el procesamiento de datos hasta que la coherencia de las banderas sea restablecido. Reflejándose en pérdida de relojes y en un incremento de la fiabilidad del sistema. Note que por consistencia con los principios de la Lógica Tortuga, las líneas de señal de los puertos de *Req* y *Ack* son duplicadas.

En la Figura 7.4 se presentan los resultados de simulación para un registro $RegH$. Se generan ruido blanco Gaussiano aditivo (AWGN) por medio de una implementación Verilog-A, con una media de 0V y una desviación estándar de 0.5V rms, utilizando diferentes semillas para cada generador de ruido, para cada dato de entrada y reloj redundante. En la Figura 7.4, son representadas de izquierda a derecha y de arriba hacia abajo como: reloj redundante (*clkt* y *clkc*), datos de entrada redundantes (*dt* y *dc*), bandera redundante que valida los datos de salida *fct* y *fcc*), señal redundante de requisición (*reqt* y *reqc*), dato de salida redundante síncrono (*qt* y *qc*) y finalmente, la señal redundante de reconocimiento (*ackt* y *ackc*), indica si el dato salida es valido o no. Cabe señalar que todas las salidas redundantes bajo prueba tienen una puerta lógica NOT TL como dispositivo de carga.

Dada la forma de como se implementa cada una de las funciones del registro $RegH$, se establecen las siguientes premisas:

- El tiempo de procesamiento, tanto de la operación secuencial redundante ($Q_t = D_t$ y $Q_c = D_c$) y la validación de la señal de reconocimiento redundante (Req_t y Req_c) son 3.4x mayores, que el tiempo de procesamiento de la bandera de salida redundante (fct y fcc).

- Toda discrepancia que ocurra en un instante de tiempo anterior al tiempo de retardo de la bandera de salida redundante (fct y fcc), será detectado y esta misma bandera redundante tendrá el tiempo suficiente para avisar a la siguiente etapa, haciendo que el procesamiento de los datos de un procesador Tortuga, deseche el dato actual, perdiendo un ciclo de reloj.

- Toda discrepancia que ocurra en un instante de tiempo posterior al tiempo de retardo de la bandera de salida redundante (fct y fcc), no será detectado por el sistema de bandera redundante, sin embargo como el tiempo de procesamiento de la operación secuencial redundante ($Q_t = D_t$ y $Q_c = D_c$) y la validación de la señal de reconocimiento redundante (Req_t y Req_c) son 3.4x mayores, que el tiempo de procesamiento de la bandera de salida redundante (fct y fcc), no afectara porque la salida de los datos secuenciales redundantes ($Q_t = D_t$ y $Q_c = D_c$) permanecerán en su valor previo que es correcto.

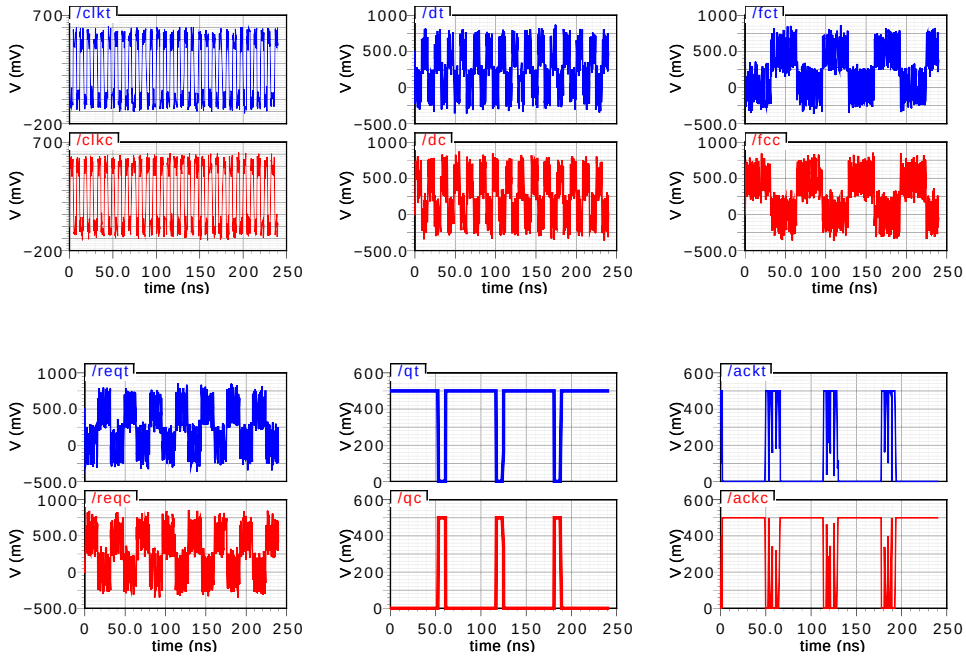


Figura 7.4: Resultados de una simulación comportamental de un registro $RegH$, bajo ruido AWGN no correlado, con una media de 0V y una desviación estándar de 0.5V rms.

Una comprobación del correcto funcionamiento del registro *RegH* es usar un espurio en una de las líneas del dato de entrada del mismo registro, que en otras palabras es un discrepancia en el dato de entrada. El espurio inyectado es caracterizado por un pulso de voltaje de amplitud V_{DD} y con un ancho de tiempo W. El espurio es inyectado de forma aleatoria respecto a un periodo de reloj T, donde, el periodo T es dividido en M rebanadas de tiempo. Por lo tanto, el tiempo de duración o ancho en tiempo W del espurio está definido como W=k·T/M, dónde k es un número natural que indica el número de rebanadas de tiempo de duración del espurio, ver Figura 7.5. Por ejemplo, para una W=5T/100, significa que el tiempo de duración del espurio tiene una duración de tiempo de 5 de 100 partes de un período T para una amplitud de voltaje de V_{DD}.

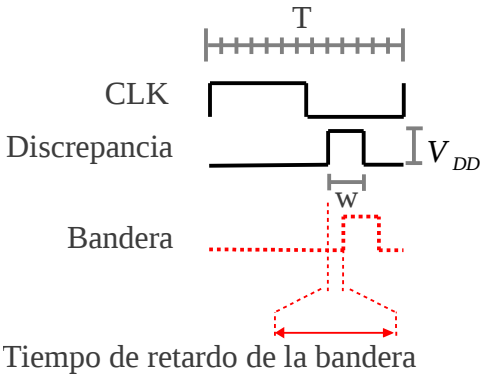


Figura 7.5: Diagrama descriptivo del concepto ruido inyectado o discrepancia inyectada.

Si la inyección del espurio se realiza en sólo una de las líneas del dato redundante, se tiene el efecto generar un efecto de discrepancia en el dato de entrada redundante del registro *RegH* con el mismo tiempo de duración W del espurio. En este caso el registro *RegH* generará una señal de bandera que indica cuando el dato de salida redundante es válido o no. En la Figura 7.5, se presenta gráficamente el concepto de discrepancia en un dato de entrada redundante, respecto a un reloj de periodo T, divido en M rebanadas con una amplitud V_{DD} y una bandera de validación. Dependiendo del tiempo en que es inyectado este espurio de efecto discrepante, se pueden tener varios casos de impacto o penalización en la operación del Registro *RegH*, los cuales son ilustrados en la Figura 7.6.

De acuerdo a las ecuaciones de comportamiento lógico el registro *RegH*, tiene características similares a los de un registro de almacenamiento temporal Flip-flop D, con las funcionalidades extras de: una señal redundante de entrada que proviene de la validación de datos de una etapa lógica combinacional Tortuga; dos señales redundantes del protocolo de comunicación

"Handshake", una de requerimiento y la otra de reconocimiento. Por lo tanto, el análisis de impacto de la inyección de un espurio al dato redundante de entrada de un registro *RegH*, se realiza como si este registro fuera un registro de almacenamiento temporal Flip-Flop D.

Un registro de almacenamiento temporal Flip-flop D esta caracterizado por sus tiempos de "Setup" y "Hold" los cuales son ilustrados en la Figura 7.6. En este caso entiéndase D1, D2, D3 y D4 como un espurio o discrepancia inyectada en el dato de entrada redundante a a lo largo del periodo T. Una vez que es inyectada una discrepancia y que el registro *RegH* detecta esta discrepancia por medio de la generación de una señal de validación o bandera, transcurre un retardo de tiempo. Esta señal de validación o bandera se representa por una señal punteada en la Figura 7.6 y tiene un tiempo de retardo que se representa por la diferencia de tiempo de la bandera respecto a la discrepancia inyectada.

- Para el caso D1 y D4 el registro *RegH* tiene suficiente tiempo para procesar esta discrepancia y generar la señal de bandera, de esta forma que por el lapso de tiempo, indicado por la señal punteada, el dato de salida procesado es invalido. Este caso no afecta al dato de salida ya que D1 y D4 ocurren antes de que se cumpla el tiempo de Setup, no afectando el tiempo de Hold y por consecuencia, no se afecta el proceso de latcheo del dato de entrada a la salida del registro *RegH*.
- Para el caso D2, la discrepancia inyectada ocurre durante el tiempo de Setup, y la señal que indica que a ocurrido una discrepancia ocurre durante el tiempo de Hold del registro *RegH*, generando que todo el sistema tenga tiempo de enterarse y por lo tanto que todo el sistema descarte ese dato y solo se pierda un ciclo de reloj.
- Para el caso D3, la discrepancia inyectada ocurre en un determinado tiempo que la señal de validación o bandera se genera una vez que ha transcurrido el tiempo de Hold. En este escenario, los elementos secuenciales que utilizan la señal de bandera de validación no tienen suficiente tiempo de enterarse de que a ocurrido una discrepancia, y por lo tanto, procesan o capturan un posible dato erróneo de salida.

Cabe hacer mención que para el caso D3, aún cuando la discrepancia ocurre en un tiempo dentro del período T qué la generación de la bandera que determina que el dato actual es incorrecto o debe despreciarse, el registro *RegH* no procesa el dato redundante de entrada, y permanece con su dato redundante correcto previo. Esto se puede ver en los resultados de simulación que se presentan en la Figura 7.7. El escenario de simulación

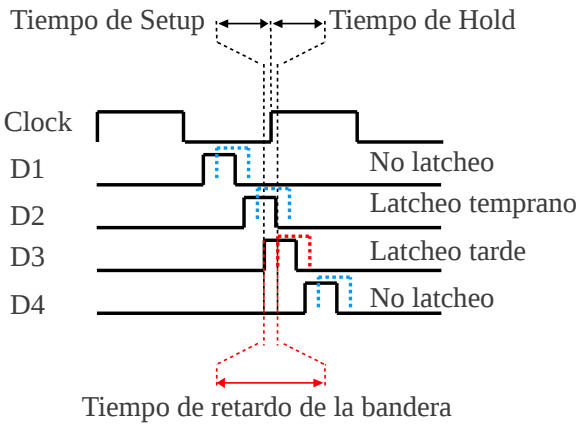


Figura 7.6: Cuatro casos genéricos de inyección de ruido o discrepancia a la entrada redundante de un registro *RegH* a lo largo del periodo T.

para el registro *RegH* se basa en realizar una inyección de una discrepancia en los datos de entrada, a lo largo del periodo T.

En la Figura 7.7, se presenta el resultado de una simulación comportamental transitoria para tres registros *RegH* conectados en cadena. El dato de salida del primer registro es el dato de entrada del segundo y así sucesivamente hasta el tercer registro, el cual tiene funcionalidad de fungir como dispositivo de cargar del segundo registro. Un espurio es inyectado en la línea verdadera del dato de entrada redundante (aot) del primer registro, a lo largo del período T, para un espurio de duración W=5T/20, con amplitud V_{DD} = 500 mV. Por simplicidad, sólo se presenta una de las líneas redundantes de todas la señales. clkt es la línea verdadera para un reloj, que es utilizado para sincronizar los tres registros *RegH*; aot y aoc son el dato verdadero y complementarios redundante de salida del primer registro y al mismo tiempo es el dato de entrada del segundo registro; aot'1 es la salida verdadera del segundo registro y finalmente aot'2 es la salida del tercer registro. De donde, se puede observar que no presenta errores, ya que el registro detecta todas las discrepancias, sin importar en dónde ocurra el espurio o discrepancia respecto a la señal de reloj, con una penalidad en ciclos de reloj.

Con el diseño y comprobación del comportamiento funcional del registro *RegH* es posible implementar el protocolo de comunicación "Handshake" Tortuga, el cual es implementado de acuerdo a la Figura 7.2, utilizando un buffer Tortuga como lógica combinacional Tortuga, obteniendo los resultados de simulación comportamental y que se presenta para un solo estado pipeline en la Figura 7.8.

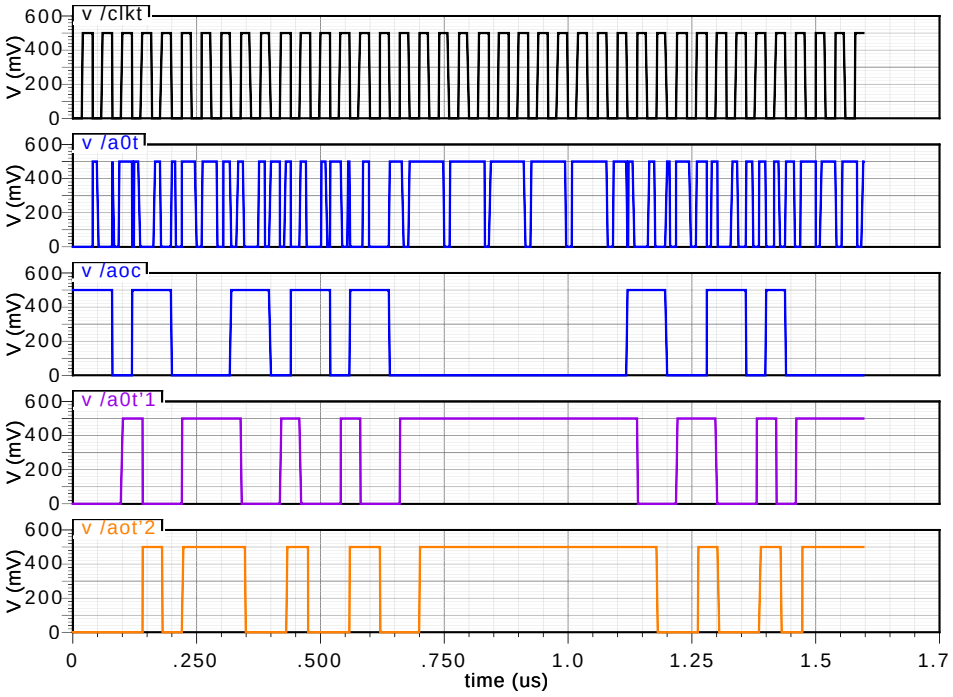


Figura 7.7: Inyección de un espurio haciendo un barrido a lo largo del periodo T, para el dato de entrada de un registro *RegH* conectado en cadena a otros tres registros similares.

Las señales que determinan el comportamiento funcional del protocolo de comunicación "Handshake", se presentan en el siguiente orden:

- d y dc: dato redundante de entrada, donde d es el la parte verdadera de dato y dc es el parte complementaria del dato.
- fc y fcc: bandera redundante de validación del dato proveniente de la Lógica Combinacional Tortuga, fc es la parte verdadera y fcc es la parte complementaria.
- req y reqc: son la señal redundante de requisición del protocolo de comunicación "Handshake", dónde req es la parte verdadera y reqc es la parte complementaria.
- clk y clkc: son la señal de reloj de todo el sistema , clk es parte verdadera y clkc es la parte complementaria del reloj.
- ack0 y ackc0: son las señales de reconocimiento del protocolo de "Handshake" y es valida cuando ack0=1 y ackc0=0. En otras palabras es el único estado que indica que se ha procesado adecuada-

mente el dato en el registro *RegH* y el procesamiento en todas las etapas pipeline se puede seguir procesando, esta señal se convierte en la señal redundante de requisición para las demás etapas pipeline. ack0 es la parte de verdadera y ackc0 es la parte complementaria de la señal de reconocimiento.

- qo y qco: son el dato redundante y complementario de salida de la etapa pipeline en análisis. Esta señal toma el valor de entrada de dato d y dc, respectivamente, si y solo si, ocurre un flaco de subida en el reloj verdadero y se cumple la parte complementaria del reloj, fc=1, fc=0, req=1 y reqc=0.

Con el diseño e implementación del Registro *RegH* mediante el uso de la Lógica Tortuga, se tienen todos los elementos tanto combinacionales como secuenciales que permiten la implementación del protocolo de comunicación "Handshake" Tortuga y por lo tanto, la implementación de un Procesador Tortuga. El cual es ejemplificado por medio de un Multiplicador que es discutido en la siguiente sección.

7.2 Multiplicador de 8X8 bits en un ambiente altamente ruidoso

Como medio de prueba y evaluación del comportamiento de la Lógica Tortuga, se utiliza un multiplicador Baugh-Wooley de 8x8-bits con signo complemento a dos [96, 97], en una estructura de diseño pipeline. Se utiliza este tipo de multiplicadores ya que es un multiplicador que se implementa mediante una estructura regular, lo que lo hace apropiado para visualizar el comportamiento de la Lógica Tortuga, por encima de una posible optimización de las prestaciones del multiplicador, cuya optimización no es el objetivo de este trabajo.

Permítanos considerar dos números enteros con signo A y B:

$$A = (a_{n-1} \cdot \cdots \cdot a_0) = -a_{n-1} \cdot 2^{n-1} + \sum_{0}^{n-2} a_i \cdot 2^i \tag{7.1}$$

$$B = (b_{n-1} \cdot \cdots \cdot b_0) = -b_{n-1} \cdot 2^{n-1} + \sum_{0}^{n-2} b_i \cdot 2^i \tag{7.2}$$

El producto de $A \cdot B$ esta dado por:

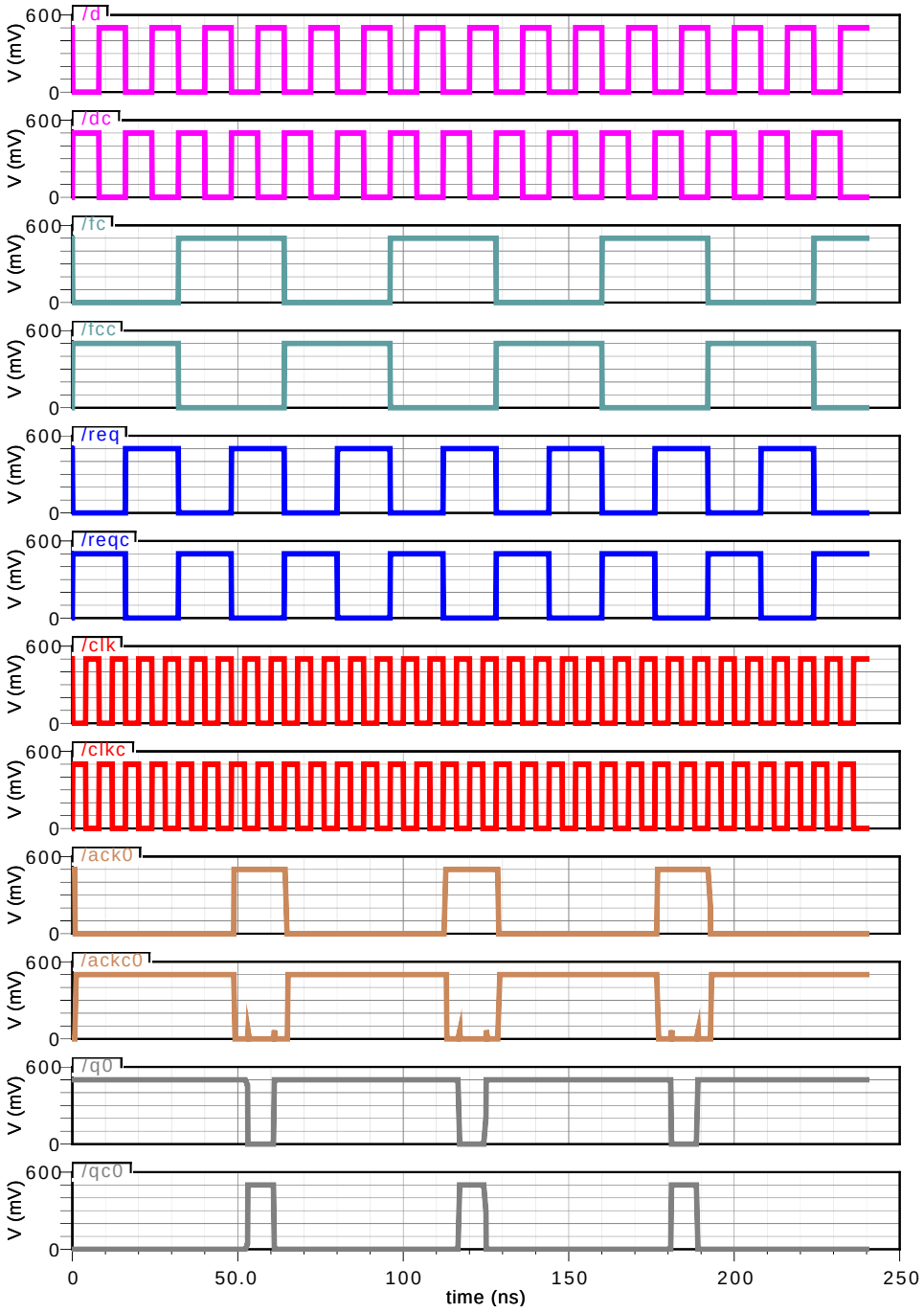


Figura 7.8: Resultados de simulación comportamental para el protocolo de Comunicación Tortuga de un solo estado pipeline.

$$A \cdot B = a_{n-1} \cdot b_{n-1} \cdot 2^{n-2} + \sum_{0}^{n-2} \sum_{0}^{n-2} a_i \cdot b_i - a_{n-1} \sum_{0}^{n-2} b_i \cdot 2^{n+i-1} - b_{n-1} \sum_{0}^{n-2} a_i \cdot 2^{n+i-1} \tag{7.3}$$

Los términos negativos se pueden reescribir como:

$$a_{n-1}\sum_{0}^{n-2} b_i \cdot 2^{n+i-1} = a_{n-1} \cdot \left(-2^{n-2} + 2^{n-1} + \sum_{0}^{n-2} \overline{b_i} \cdot 2^{n+i-1} \right) \tag{7.4}$$

Usando esta aproximación, $A \cdot B$ puede escribirse como:

$$\begin{aligned} A \cdot B &= a_{n-1} \cdot b_{n-1} \cdot 2^{n-2} + \sum_{0}^{n-2}\sum_{0}^{n-2} a_i \cdot b_i \cdot 2^{n-2} \\ &+ b_{n-1}\left(-2^{n-2} + 2^{n-1} + \sum_{0}^{n-2} \overline{a_i} \cdot 2^{n+i-1} \right) \\ &+ a_{n-1}\left(-2^{n-2} + 2^{n-1} + \sum_{0}^{n-2} \overline{b_i} \cdot 2^{n+i-1} \right) \end{aligned} \tag{7.5}$$

La ecuación anterior puede expresarse de una forma más conveniente por medio de la igualdad siguiente:

$$-\left(b^{n-1} + a^{n-1}\right) \cdot 2^{2n-2} = -2^{2n-1} + -\left(\overline{a_{n-1}} + \overline{b_{n-1}} \cdot 2^{2n-2}\right) \tag{7.6}$$

Por lo tanto, $A \cdot B$ esta dada por:

$$\begin{aligned} A \cdot B &= 2^{2n-1} + \left(\overline{a_{n-1}} + \overline{b_{n-1}} + a^{n-1} \cdot b^{n-1}\right) \cdot 2^{2n-2} \\ &+ \sum_{0}^{n-2}\sum_{0}^{n-2} a_i \cdot b_j \cdot 2^{i+j} + (a_{n-1} + b_{n-1}) \cdot 2^{n-1} \\ &+ \sum_{0}^{n-2} b_{n-1} \cdot \overline{a_i} \cdot 2^{n+1-j} + \sum_{0}^{n-2} a_{n-1} \cdot \overline{b_i} 2^{n+i-1} \end{aligned} \tag{7.7}$$

Ya que A y B son operandos de n-bits, sus productos pueden ser extendidos a $2n$-bits. El bit más significativo es tomado como -2^{2n-1}, el cual es alimentado como un 1 lógico en la celda más significativa del multiplicador Baugh-Wooley de 8x8-bits con signo, complemento a dos [96, 97].

El multiplicador Baugh-Wooley puede ser implementado por medio de una estructura tipo arreglo, el cual permite una implementación lineal mediante una alimentación de datos hacia adelante y por lo tanto, es directo realizar una estructura pipeline. El circuito del multiplicador es implementado por medio de cuatro estados pipeline. Cada estado pipeline procesa

4 bits de la operación de multiplicación. Dos metodologías son probadas mediante el multiplicador Baugh-Wooley, la convencional y la segunda mediante los principios de la Lógica Tortuga. Los multiplicadores convencional y Tortuga son implementados utilizando:

- Dos procesadores de ayuda, el primero suministra los datos de entrada al multiplicador bajo prueba, y el segundo es el que recibe los datos procesados por el mismo multiplicador.
- Un protocolo de comunicación "Handshake", basado en dos señales de control, requisición y reconocimiento, analizado en una sección previa.
- Cuatro estados pipeline, en donde cada estado procesa 4 bits de la operación de multiplicación.
- Tecnología CMOS a 90nm.

Como experimentos de demostración y comparación de los multiplicadores implementados mediante una técnica convencional y la lógica Tortuga se realizan dos experimentos de simulación. Los experimentos se realizan a ambos multiplicadores ya que permite una comparación directa en términos de fiabilidad. En el primer experimento se inyectan espurios a las líneas primarias de entrada de los dos multiplicadores. En el segundo experimento se inyecta ruido aleatorio en los nodos internos de la segunda etapa pipeline de ambos multiplicadores. De donde, en la Figura 7.9, se ilustrada por medio de circunferencias el lugar de la inyección de los espurios en las líneas del bit menos y más significativo, así como del primer como del segundo nivel "Pipeline" de ambos multiplicadores.

El espurio aplicado a ambos experimentos es caracterizado por una discrepancia de datos de amplitud V_{DD}, con un ancho W localizado de forma aleatoria sobre la señal redundante de reloj definido en un periodo T. El periodo T es dividido en M rebanadas de tiempo y la discrepancia es definida por W=k·T/M, dónde k es un número natural que indica el número de rebanadas de duración de la discrepancia (Figura 7.6). Por ejemplo 5T/20, significa que la discrepancia tiene una duración de 5/20 partes del período T.

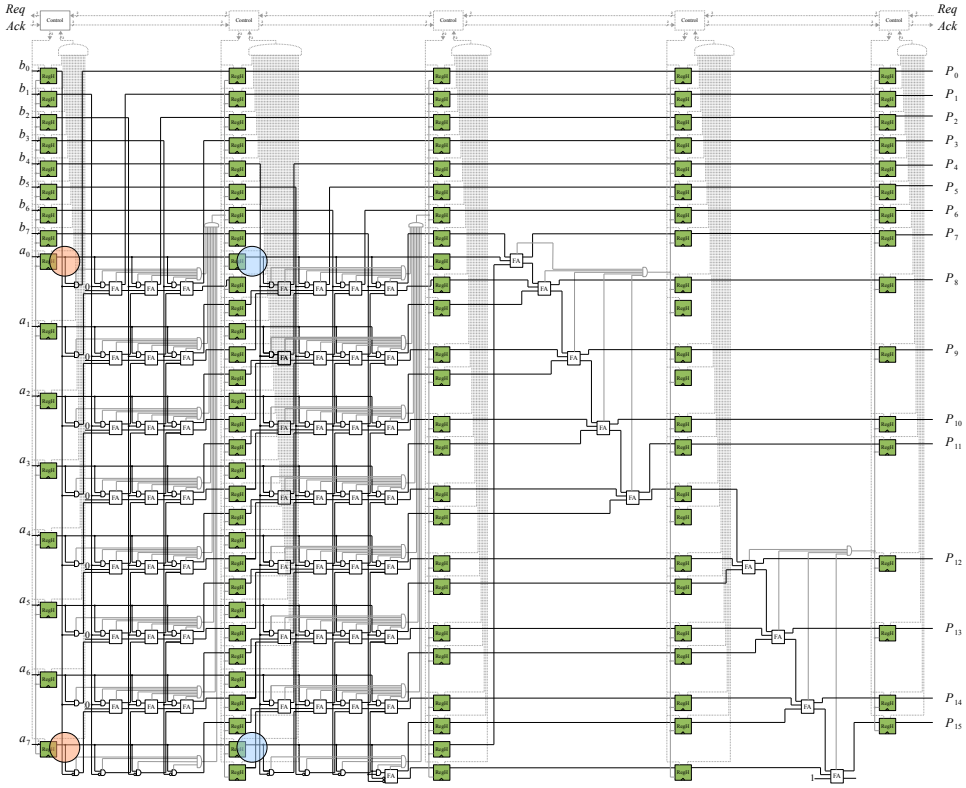


Figura 7.9: En la figura se muestran los lugares donde es inyectado el ruido (espurios) para un multiplicador. En el experimento 1, la inyección de ruido es aplicado en los nodos de las entradas primarias, mientras que en el experimento 2 es aplicado en los nodos internos del segundo estado pipeline.

7.3 Experimento 1: Ruido en los nodos de entrada primarios

Se inyectan espurios de manera aleatoria que generan discrepancias con una distribución equiprobable dentro de cada rebanada de tiempo del período T. Varios anchos de discrepancia son considerados en el experimento. Cada experimento implica 1,000 muestras discrepantes simples para ambos multiplicadores y sólo para el multiplicador Tortuga, 1000 muestras discrepantes dobles. El tiempo inicial de cada discrepancia simple y doble, esta localizado de forma aleatoria dentro del periodo T. Las discrepancias inyectadas en el bit menos significativo (LSB) y en el bit más significativo (MSB) son independientes y en instantes de tiempo distintos. Para el caso del multiplicador Tortuga son considerados dos diferentes casos; en el primer caso la discrepancia es inyectada en sólo una de las líneas (línea verdadera del dato)

y en el segundo caso la discrepancia es inyectada en ambas líneas del dato. Cada discrepancia es generada con el mismo principio de aleatoriedad y con independencia probabilística, por lo tanto, las señales discrepantes dobles son no correladas.

Los resultados de una simulación transitoria funcional para un multiplicador convencional con la estructura de la Figura 7.9, son mostrados en las Figuras 7.10 y 7.11. De dónde, clk es la señal global de reloj; a0, a1, a2 y a3 son los datos de entrada A y B, por medio de la siguiente relación A=(a0 a1 a2 a3 a0 a1 a2 a3) y B=(a3 a2 a1 a0 a3 a2 a1 a0); y el producto de la multiplicación AxB esta determinada por los bits p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12 p13 p14 y p15.

La comparación de resultados del comportamiento de ambos multiplicadores, convencional y Tortuga para una línea simple de discrepancia y discrepancia doble en las entradas primarias del bit menos significativo LSB así como en el más significativo son mostrados en la Tabla 7.1. Los resultados corresponden a un experimento con 1,000 muestras de discrepancias aleatorias con diferentes anchos. En todos los casos el período de trabajo o el tiempo de duración de un dato es T=15ns.

La Tabla 7.1 muestra los resultados de los experimento 1, donde podemos ver que una implementación basada en la lógica Tortuga experimenta cero errores a la salida del multiplicador en el caso de una discrepancia simple. Cuando el mecanismo Tortuga detecta un error causado por una discrepancia, este asume que la salida en ese instante de tiempo no es valida y por medio del protocolo de "Handshake", así como de las lógicas combinacional y secuencial Tortuga, el procesamiento de los datos es detenido por un ciclo de reloj. Cabe resaltar que aún cuando el multiplicador Tortuga presenta cero errores a la salida para este experimento, la penalización de la lógica Tortuga es que hay un número significativo de datos repetidos o relojes perdidos. Cada reloj perdido indica una discrepancia simple detectada. Para las mismas características de ruido el multiplicador convencional presenta una relación importante entre un 6% y un 47% de errores.

La tabla 7.1 presenta la relación de errores cuando las discrepancias afectan ambas líneas de los respectivos nodos de entrada, señal verdadera y complementaria con generación independiente una respecto a la otra, mostrando para este experimento un máximo de errores de 0.2%.

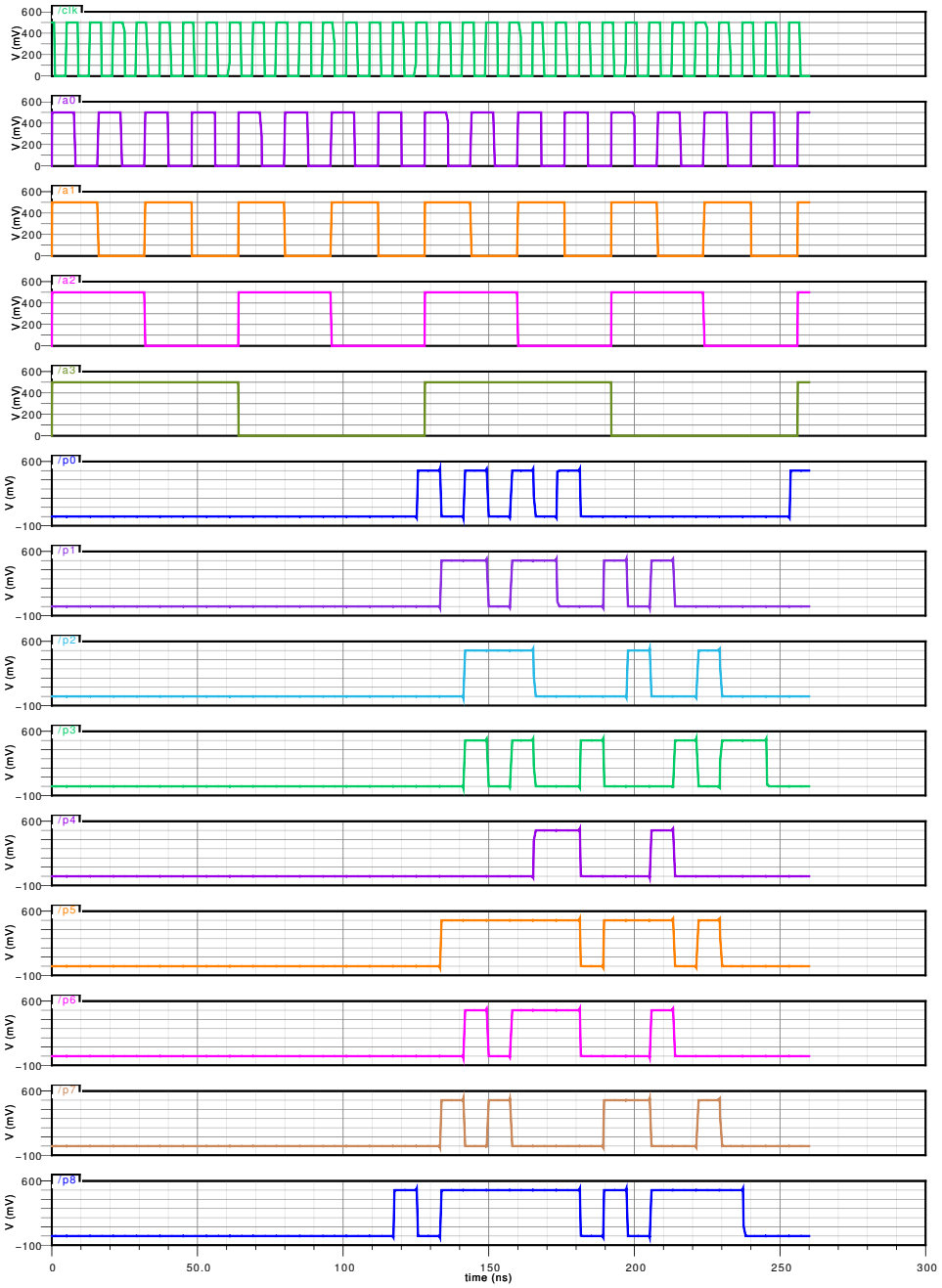


Figura 7.10: En la figura se muestran los resultados de una simulación transitoria para un multiplicador convencional de 8x8 bits con signo, utilizando 4 etapas pipeline.

7.4 Experimento 2: Ruido en nodos internos

En esta sección se aplica el mismo experimento de la sección anterior inyectando ruido en nodos internos del multiplicador tanto para de forma

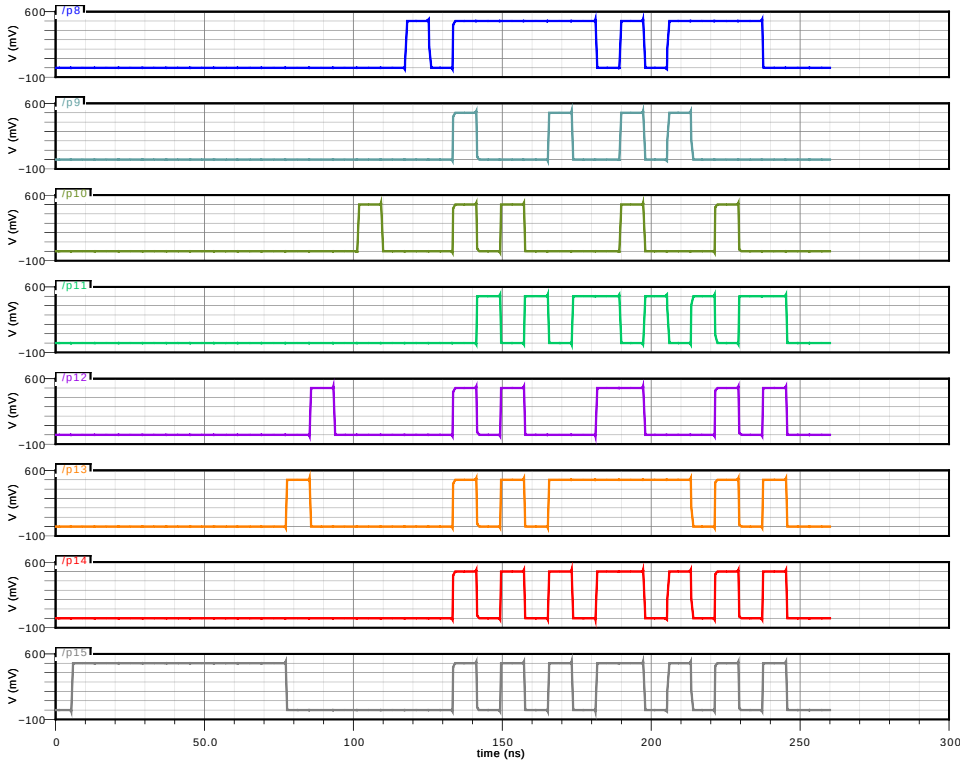


Figura 7.11: En la Figura se muestran los resultados de los bits más significativos de una simulación transitoria, para un multiplicador convencional de 8x8 bits con signo, utilizando 4 etapas pipeline.

simple como de forma diferencial (tanto en el nodo verdadero como en el complementario) de las puertas correspondientes a las rutas LSB y MSB del segundo estado pipeline del multiplicador TL.

La Tabla 7.2 muestra los resultados del experimento para una discrepancia con diferentes anchos de tiempo, utilizando 1,000 datos para cada caso. Análogamente para los resultados del experimento 1, cuando el ruido es aplicado en sólo las líneas verdaderas de los estados internos, el multiplicador Tortuga se recupera de todos las discrepancias. Nótese que el costo en ciclos de reloj perdidos ahora es menor porque la ruta de retardo es más pequeña ya que el ruido ha sido introducido en un estado intermedio. Para discrepancias dobles inyectadas en ambas líneas de nodos verdadera y complementaria, la relación de error es también pequeña la cual se encuentra en un rango de menor de 0.9%.

7.5 Conclusiones

Un severo modelo de ruido ha sido usado en los experimentos aplicado a una implementación de un multiplicador 8x8-bits pipelined Baugh-Wooley complemento a dos. Los experimentos revelan una tolerancia perfecta para el caso de discrepancias en líneas sencillas (sin errores a la salida de todos los casos) para ambos nodos primarios e internos con un costo de perdida de relojes, entre un 6% y un 25% para el ruido inyectado en los experimentos. Esto significa que las prestaciones del sistema disminuyen sin embargo mantiene un alto nivel de fiabilidad.

Para el mismo experimento, un multiplicador convencional exhibe una alta relación de error entre un 6% y un 48%. La relación de error para la implementación lógica Tortuga propuesta con doble discrepancia en ambas líneas verdadera y complementaria, es menor que el 0.1% cuando el ruido afecta nodos de entrada primarios y es menor al 0.9% cuando el ruido afecta a nodos internos. Sólo discrepancias simultaneas para ambas líneas pueden generar un error.

Tabla 7.1: Resultados para los casos de discrepancia sencilla y doble inyectadas en los líneas primarias de entrada, para 1,000 muestras y un periodo $T = 15ns$.

Multiplicadores Convencional and Lógica Tortuga										
Ancho de ruido	Discrepancias inyectadas en el bit LSB					Discrepancias inyectadas en el bit MSB				
	Conv.	Multiplicador TL				Conv.	Multiplicador TL			
	Errores	Discrepancia simple		Doble discrepancia		Errors	Discrepancia simple		Doble discrepancia	
		LC	Errores (%)	RP (%)	Errores (%)		RP (%)	Errores (%)	RP (%)	Errores (%)
3T/50	16.6	11.4	0	11.9	0.1	17.6	9.1	0	13.6	0.0
5T/50	23.8	17.4	0	23.5	0.0	24.0	14.5	0	22.0	0.0
7T/50	31.4	23.4	0	31.8	0.0	30.2	21.3	0	32.2	0.1
10T/50	39.2	32.7	0	47.6	0.0	19.0	33.0	0	47.6	0.0
12T/50	47.6	40.9	0	62.5	0.1	48.0	41.4	0	61.2	0.0
5T/100	6.4	6.2	0	6.3	0.1	6.5	5.7	0	6.6	0.1
10T/100	12.6	10.7	0	16.7	0.0	12.7	10.0	0	15.1	0.0
15T/100	17.7	13.6	0	23.1	0.1	18.2	17.0	0	23.2	0.2
20T/100	23.0	18.5	0	30.7	0.0	24.3	19.6	0	28.6	0.0
25T/100	28.1	23.2	0	39.8	0.0	28.5	25.4	0	37.8	0.0
(Conv.) Multiplicador convencional y (RP) Relojes perdidos										

Tabla 7.2: Resultados para los casos de discrepancias sencilla y doble inyectadas en nodos de datos internos, 1,000 muestras son procesadas en un periodo de $T = 15ns$.

Multiplicador TL								
Ancho de ruido	Discrepancias inyectadas en LSB				Discrepancias inyectadas en MSB			
	Discrepancia simple		Discrepancia doble		Discrepancia simple		Discrepancia doble	
	LC (%)	Errores (%)	RP (%)	Errores (%)	RP (%)	Errores (%)	RP (%)	Errores (%)
5T/100	5.7	0	6.0	0.6	6.1	0	6.4	0.7
10T/100	6.3	0	15.2	0.9	6.5	0	14.5	0.6
15T/100	10.8	0	22.9	0.3	11.5	0	22.0	0.1
20T/100	15.9	0	29.5	0.8	15.6	0	25.1	0.7
25T/100	23.1	0	38.1	0.5	24.0	0	27.6	0.4
(RP) Relojs perdidos.								

Capítulo 8

Conclusiones

En escenarios futuros de baja potencia y bajo voltaje los sistemas electrónicos presentaran una alta relación de errores suaves o transitorios debido a una drástica relación señal a ruido. Estos errores transitorios pueden afectar los resultados lógicos en un forma permanente. En esta tesis, se ha mostrado una nueva lógica basada en múltiples líneas redundantes para cada nodo lógico como una alternativa para las estrategias basadas en triple redundancia (TMR) dentro de un escenario tolerante a fallas.

La distribución de probabilidades de voltajes en un nodo digital ruidoso, puede ser descrito como la unión de dos distribuciones Gaussianas centradas alrededor del 0-lógico y 1-lógico, que en términos de tensión son 0 y V_{DD}, respectivamente.

Un circuito digital presenta un error cuando un valor lógico es mal interpretado, es decir, cuando un 1-lógico es interpretado como un 0-lógico y viceversa, esto esta definido por la probabilidad de error P_e.

Una posible forma de reducir la probabilidad de error para un nodo digital ruidoso es por medio del incremento del número de puertos (redundancia de puertos (PR-N)), de tal manera que la probabilidad de error se reduce a medida que se incrementa el número de puertos.

En un escenario con un nivel muy alto de ruido, por ejemplo para un SNR de -3dB, la lógica convencional presenta una perdida de fiabilidad aproximada del 50%.

Con base en los análisis de fiabilidad previos, en este trabajo de investi-

gación se presenta un nuevo paradigma de diseño nombrado Lógica Tortuga, el cual se basa en la redundancia complementaria de líneas para cada nodo lógico.

La probabilidad de error para una puerta lógica NOT convencional sin replica de puertos (PR-0), en un escenario con un SNR=-3dB ($\sigma = 0.5$ (Vrms) y $V_{DD} = 0.5$ (V)) es del 35%, mientras que para una redundancia de PR-1 la probabilidad de error es de alrededor del 12%, lo que significa una mejoria de la fiabilidad del 65% respecto a PR-0. Cuando PR es incrementada a 2, 3 y 4 veces, las correspondientes probabilidades de error se reducen a 11.95%, 4.13%, 1.41%, 4.91e-3%, respectivamente.

Cuando PR va más allá de 4 veces un decremento insignificante es observado con la desventaja del incremento en los requerimientos de hardware para implementar el circuito con la misma funcionalidad y prestaciones deseadas.

El coste en hardware respecto al incremento de la redundancia de PR en un circuito digital es inversamente proporcional.

Por lo tanto y con base al análisis anterior, una redundancia de PR-1 es considerado en este trabajo, ya que es un buen balance entre fiabilidad y coste en hardware.

La nueva lógica es apropiada para detectar y detener la propagación de las discrepancias (errores simples). El uso de doble línea o dato (dos líneas para cada entrada y cada salida) disminuye la probabilidad de error cuando el ruido causa discrepancias sobre ambas líneas.

Los resultados presentados en esta sección demuestran que la Lógica Tortuga, propuesta en esta tesis a nivel puerta lógica, presenta una excelente tolerancia a señales espurias respecto a las metodologías convencional o estática y las basadas en Markov Random Field (MRF), como se presenta en la Tabla 4.3., donde se procesan 1,000 datos con señales espurias, 5,000 unos lógicos y 5,000 ceros lógicos. Con un voltaje de alimentación V_{DD}=0.15 V y una temperatura de 100^0C. De donde la Lógica Tortuga para un diseño de puerta lógica NOT y buffer es aproximadamente 30.6X y 116.8X veces más tolerante a señales espurias que las lógicas MRF y estática CMOS, respectivamente. Realizando un comparativa respecto a las puertas AND2, NAND2, OR2, NOR2, XOR2 y XNOR2, se obtiene que la Lógica Tortuga es 9.7X, 11.6X, 4.4X, 11.5X, 7.3X y 7.8X más tolerante a señales espurias que la Lógica estándar CMOS.

El ruido generado para cada transistor y para cada fuente de alimentación

es amplificado 100 veces respecto al ruido intrínseco para una tecnología de 90nm, para emular el ambiente de tecnologías futuras que tendrán componentes muy ruidosos. Cabe señalar que el voltaje nominal para una tecnología de 90nm es de 1V, pero en esta tesis se decremento a 0.15V para simular las condiciones de trabajo de tecnologías futuras.

La lógica Tortuga presenta una penalización en hardware la cual se ve refleja en un coste de hardware de 1X, 5X, 10X mayor que las lógicas basadas en MRF, DCVS y estándar CMOS, para una puerta lógica NOT. De 0.5X, 3.2X y 8X para una puerta Lógica NAND. De 0.5X, 3.2X y 8X para una puerta Lógica NOR y finalmente de 3.1X, 0.9X y de 2X para una puerta Lógica XOR.

Una posible forma de medir la inmunidad al ruido de un elemento lógico es mediante la distancia de Kullback-Leibler, la cual mide la discrepancia entre una distribución de probabilidades de una salida real respecto a una distribución de probabilidades ideal. En un sistema digital donde se tienen dos niveles lógicos, la distancia de Kullback-Leibler mide la distancia entra la distribución de probabilidades de la salida real respecto a la distribución de probabilidad de una señal ideal. Entre más pequeña es la medida de Kullback-Leibler es mayor la inmunidad a ruido de un elemento digital. De acuerdo los resultados presentados en la Tabla 4.5 de las medidas de Kullback-Leibler para un diseño estándar CMOS, uno basado en Differential Cascode Voltaje Swith (DCVS), uno basado en Markov Random Field (MRF) y uno en la Lógica Tortuga, la lógica Tortuga es aproximadamente 3.6X, 13.4X y 20.9X veces mejor que MRF, DCVS y estándar CMOS, respectivamente.

Del análisis de errores para un sumador completo utilizando la Lógica Tortuga se establece que un sistema de banderas que indique cuando una dato de salida lógico es valido o no, es necesario.

Se han implementado los elementos secuenciales básicos Latch y Flip-Flop D, por medio de la Lógica Tortuga.

Estos elementos secuenciales basados en la Lógica Tortuga heredan las bondades de la lógica tortuga en términos de inmunidad al ruido, así como penalizaciones en tiempo y hardware.

Realizando una comparación de la tolerancia al ruido para un Flip-Flop D, por medio de la Distancia de Kullback-Leibler, que determina la discrepancia entre la distribución de probabilidades de la salida ruidosa respecto a la distribución de probabilidades de la salida ideal, se obtuvo que la lógica Tortuga es 2.3X, 4.6X y 6.1X, mejor que las técnicas MRF reinforcer, 1st-

order CTMR y TMR, respectivamente.

Un severo modelo de ruido ha sido usado en los experimentos aplicado a una implementación de un multiplicador 8x8-bits pipelined Baugh-wooley complemento a dos. Los experimentos revelan una tolerancia perfecta para el caso de discrepancias en líneas sencillas (sin errores a la salida de todos los casos) sin embargo para el caso de ambos nodos primarios e internos existe una penalización de perdida de relojes, entre un 6% y un 25%. Esto significa que las prestaciones del sistema disminuyen sin embargo la lógica Tortuga mantiene un alto nivel de fiabilidad.

Para el mismo experimento, un multiplicador convencional exhibe una alta relación de error entre un 6% y un 48%. La relación de error para la implementación Lógica Tortuga propuesta con doble discrepancia en ambas líneas verdadera y complementaria, es menor que el 0.1% cuando el ruido afecta nodos de entrada primarios y es menor al 0.9% cuando el ruido afecta a nodos internos. Sólo discrepancias simultaneas para ambas líneas pueden generar un error.

Appendix A

Implementación de alto nivel: Verilog-A

A.1 Procesador1 para los multiplicadores convencional y Tortuga implementado mediante una descripción Verilog-A

```
// Author: Lancelot Garcia Leyva, Ph.D student
// México: Universidad Autónoma de Tlaxcala
// España: Universitat Politècnica de Catalutunya
// it determine de behavior
// and the measurements of the turtle multiplier
// transmiter turtle handshake logic
// $Date: 2010/06/14$
// $Revision: 1.0 $
//
// INSTANCE parameters
// vlogic_high = output voltage for high [V]
// vlogic_low = output voltage for high [V]
// vtrans = voltages above this at input are considered high [V]
// tdel, trise, tfall = {usual} [s]
//

#include "discipline.h"
#include "constants.h"
```

```
module transmitter_ss(clkt, clkc, at0, ac0, at1, ac1, at2, ac2,
at3, ac3, at4, ac4, at5, ac5, at6, ac6, at7, ac7, bt0, bc0,
bt1, bc1, bt2, bc2, bt3, bc3, bt4, bc4, bt5, bc5,
bt6, bc6, bt7, bc7, reqt, reqc, ackt, ackc, flag_sample);
input ackt, ackc;
output clkt, clkc;
output at0, ac0, at1, ac1, at2, ac2, at3, ac3,
at4, ac4, at5, ac5, at6, ac6, at7, ac7;
output bt0, bc0, bt1, bc1, bt2, bc2, bt3, bc3,
bt4, bc4, bt5, bc5, bt6, bc6, bt7, bc7;
output reqt, reqc;
output flag_sample;

electrical ackt, ackc, reqt, reqc, clkt, clkc,
flag_sample;
electrical at0, ac0, at1, ac1, at2, ac2, at3,
ac3, at4, ac4, at5, ac5, at6, ac6, at7, ac7;
electrical bt0, bc0, bt1, bc1, bt2, bc2, bt3,
bc3, bt4, bc4, bt5, bc5, bt6, bc6, bt7, bc7;

integer xt, xc, cont_flag_sample;
integer dat0, dac0, dat1, dac1, dat2, dac2, dat3,
dac3, dat4, dac4, dat5, dac5, dat6, dac6, dat7, dac7;
integer dbt0, dbc0, dbt1, dbc1, dbt2, dbc2, dbt3,
dbc3, dbt4, dbc4, dbt5, dbc5, dbt6, dbc6, dbt7, dbc7;
integer logica_ackt, logica_ackc, logicareqt, logicareqc;
integer cont_T, i0, cont_clock, seed, ss, bandera_ss;
integer dec_A, dec_B, dec_AB, lost_clocks, data_return;
integer fileID;
// variable to store the pointer to a file
real var_T;

parameter num_steps=100;
// is the value which divides the period and generate SS
parameter dat_ss=num_steps/4;
// this valor determine the time when the ss is generated
// and go back the correct value.
// for example dat_ss=5=num_steps/4
parameter period = 15n;    //16n
parameter sample = period/num_steps;
parameter real trise = 0.1n from (0:inf);
parameter real tfall = 0.1n from (0:inf);
  parameter real vdd_t = 0.5;  // Power supply voltage.
parameter real vlogic_high = vdd_t;
```

```
parameter real vlogic_low = 0;
parameter real vtrans = vdd_t/2;  //threshold voltage



////////////////////////////////////////////////////
///////////////varibles are initialized //////////////

   analog begin

@ ( initial_step ) begin
  //internal variables to generate clock signals
  xt=0; xc=1;
  // internal varibles to generate the data A and B for multiplication
  dat0=0; dac0=1;
  dat1=0; dac1=1;
  dat2=0; dac2=1;
  dat3=0; dac3=1;
  dat4=0; dac4=1;
  dat5=0; dac5=1;
  dat6=0; dac6=1;
  dat7=0; dac7=1;
  dbt0=0; dbc0=1;
  dbt1=0; dbc1=1;
  dbt2=0; dbc2=1;
  dbt3=0; dbc3=1;
  dbt4=0; dbc4=1;
  dbt5=0; dbc5=1;
  dbt6=0; dbc6=1;
  dbt7=0; dbc7=1;
  logicareqt=1; logicareqc=0;
  bandera_ss=10;
  cont_flag_sample=0;
  data_return=dat_ss+1;
  cont_T=0;
  ss=65;
  lost_clocks=0;
  // ojo for ensure the syncrony of datas.
  cont_clock=num_steps/2;
  // se usan para llevar la cuenta de los datos en formato clock.
  i0=num_steps;
  // es la semilla que se utiliza para obtener valores aleatorios.
  seed = 4;
  // es el archivo donde se guardan los datos de la multiplicación
```

```
  // archivo con separadores
  fileID = $fopen("/homespin/users/lgleyva/transmitter_ss.csv");
  // se guardan en el archivo las cabeceras que indican
  // el orden de almacenamiento de los datos
  $fstrobe(fileID,"repeated_data_flag,A(binary),B(binary),A(dec),
  B(dec),A*B(dec),SS(0-nums_steps),cont_T,var_T");
end

////////////////////////////////////////////////////////////
// se asignan valores de entrada a las variables
// internas logica_ackt y logica_ackc
// son las variables que indican el reconocimiento
// del dato del modulo turtle.
logica_ackt = V(ackt) > vtrans;
logica_ackc = V(ackc) > vtrans;
      @ (cross(V(ackt) - vtrans, 1))  logica_ackt = 1;
      @ (cross(V(ackt) - vtrans, -1))  logica_ackt = 0;
      @ (cross(V(ackc) - vtrans, 1))  logica_ackc = 1;
      @ (cross(V(ackc) - vtrans, -1))  logica_ackc = 0;


////////////////////////////////////////////
/// Se generan clkt and clkc ///////////////
/// Se aplica reset a reqt and reqc ////////
@ (timer(0, sample)) begin
// se invierte el valor de cont_flag_sample,
// que genera un reloj 20 veces mas rápido
// que el clock general para realizar la sincronización entre
// el transmitter_ss y los receiver_ss y receiver_conventional.
cont_flag_sample=1-cont_flag_sample;
//se verifica el cont_clock para determinar T/2 e invertir el clock,
//de esta manera se genera el clock.
if (cont_clock == (num_steps/2)) begin
  cont_clock=1;
  // Clock free run
  xt=1-xt; xc=1-xc;
end
else begin
cont_clock=cont_clock+1;
end

//i0 is the counter which has the number of samples by period,
//therefore if i0=num_steps is generated a new values.
if (i0 == num_steps)begin
```

```
if (!logica_ackc && logica_ackt)begin
  // se generan los datos aleatorios para A(0:7)
  dat0=$rdist_uniform(seed,0,1); dac0=1-dat0;
  dat1=$rdist_uniform(seed,0,1); dac1=1-dat1;
  dat2=$rdist_uniform(seed,0,1); dac2=1-dat2;
  dat3=$rdist_uniform(seed,0,1); dac3=1-dat3;
  dat4=$rdist_uniform(seed,0,1); dac4=1-dat4;
  dat5=$rdist_uniform(seed,0,1); dac5=1-dat5;
  dat6=$rdist_uniform(seed,0,1); dac6=1-dat6;
  dat7=$rdist_uniform(seed,0,1); dac7=1-dat7;
  // B(0) to B(7) bit-data are generated using a random function.
  dbt0=$rdist_uniform(seed,0,1); dbc0=1-dbt0;
  dbt1=$rdist_uniform(seed,0,1); dbc1=1-dbt1;
  dbt2=$rdist_uniform(seed,0,1); dbc2=1-dbt2;
  dbt3=$rdist_uniform(seed,0,1); dbc3=1-dbt3;
  dbt4=$rdist_uniform(seed,0,1); dbc4=1-dbt4;
  dbt5=$rdist_uniform(seed,0,1); dbc5=1-dbt5;
  dbt6=$rdist_uniform(seed,0,1); dbc6=1-dbt6;
  dbt7=$rdist_uniform(seed,0,1); dbc7=1-dbt7;
  // "cont_T" is the general variable that containt
  // the valor of number of periods equal to procesing datas.
  cont_T=cont_T+1;
  // Multiplication is checking both decimal and binary
  $strobe("A=%d%d%d%d%d%d%d%d",dat7,dat6,dat5,
  dat4,dat3,dat2,dat1,dat0);
  $fstrobe(fptr,"A=%d%d%d%d%d%d%d%d",dat7,dat6,
  dat5,dat4,dat3,dat2,dat1,dat0);
  // "A" positive or negative is determinated (dat7==1)
  if (dat7==1)begin
  dec_A=(dat0+(dat1*2)+(dat2*4)+(dat3*8)+(dat4*16)+
  (dat5*32)+(dat6*64))-128;
  end
  else begin
  dec_A=dat0+(dat1*2)+(dat2*4)+(dat3*8)+(dat4*16)+
  (dat5*32)+(dat6*64)+(dat7*128);
  end
  $strobe("B=%d%d%d%d%d%d%d%d",dbt7,dbt6,dbt5,dbt4,
  dbt3,dbt2,dbt1,dbt0);
  //$fstrobe(fptr,"B=%d%d%d%d%d%d%d%d",dbt7,dbt6,dbt5,
  dbt4,dbt3,dbt2,dbt1,dbt0);
  // se determina si "B" es positivo o negativo (dbt7==1)
  if (dbt7==1)begin
    dec_B=(dbt0+dbt1*2+dbt2*4+dbt3*8+dbt4*16+dbt5*32+dbt6*64)-128;
  end
```

```
    else begin
       dec_B=dbt0+dbt1*2+dbt2*4+dbt3*8+dbt4*16+dbt5*32+dbt6*64+dbt7*128;
    end
    dec_AB=dec_A*dec_B;
     // Result of multiplication operation is shown on screen
    $strobe("%d * %d = %d",dec_A,dec_B,dec_AB) ;
    // Result of multiplication is store in a file
    $fstrobe(fptr,"%d * %d = %d",dec_A, dec_B, dec_AB);

    ////////////////////////////////////////////////////////
    // primero se decide aleatoriamente si se genera o no un SS
    // solo en en el caso de que se haya determinado generar un SS
    // entonces se genera aleatoriamente un tiempo para generar
    // el SS dentro de T=periodo
    //bandera_ss=$rdist_uniform(seed,0,1);
    //if (bandera_ss==0)
    //begin
    bandera_ss=0;
    //ss=$rdist_uniform(seed,0,num_steps-1);
    ss=ss+1;
    if (ss==num_steps)begin
    ss=0;
    end
    var_T=(period*cont_T)+(ss*sample);
    $strobe("SS in the time %d del T%d", ss, cont_T);
    $fstrobe(fileID,"0,%d%d%d%d%d%d%d%d,%d%d%d%d%d%d%d%d,%d,%d,
    %d,%d,%d,%g",dbt7,dbt6,dbt5,dbt4,dbt3,dbt2,dbt1,dbt0,dat7,
    dat6,dat5,dat4,dat3,dat2,dat1,dat0,dec_A,dec_B,dec_AB,
    (ss*10)/2,cont_T,var_T);
    // end of SS function
  end
  else begin
    lost_clocks=lost_clocks+1;
    $strobe("Clock lost %d del T%d", ss, cont_T);
    $fstrobe(fileID,"1,%d%d%d%d%d%d%d%d,%d%d%d%d%d%d%d%d,%d,%d,
    %d,%d,%d,%g",dbt7,dbt6,dbt5,dbt4,dbt3,dbt2,dbt1,dbt0,dat7,
    dat6,dat5,dat4,dat3,dat2,dat1,dat0,dec_A,dec_B,dec_AB,
    (ss*10)/2,cont_T,var_T);
  end // finaliza if (!logica_ackc && logica_ackt)
  i0=0;
end // finaliza el if(i0==num_steps)
i0=i0+1;
// The SS is genetared using three signal for control de trigger timer,
// the time is random using a normal distribution with the same likelihood
```

```
if (i0==ss && bandera_ss==0)begin
  dat7=1-dat7;
  data_return=-1;
end
// dat_ss=num_steps/4 or scaning is the width-time of
// SS that go since 0 to 25% of T, in other words (T/4) of the period.
if  (data_return==dat_ss && bandera_ss==0)begin
  dat7=1-dat7;
  bandera_ss=1;
  data_return=dat_ss+1;
end
if (data_return<num_steps)begin
  data_return=data_return+1;
end
      end

// One time the values of the clock and
// the datas are obtained are asigned to the output node.

V(clkt) <+ transition (xt*vdd_t, 0.0, trise, tfall );
V(clkc) <+ transition (xc*vdd_t, 0.0, trise, tfall );
V(reqt) <+ transition (logicareqt*vdd_t, 0.0, trise, tfall );
V(reqc) <+ transition (logicareqc*vdd_t, 0.0, trise, tfall );
V(at0) <+ transition (dat0*vdd_t, 0.0, trise, tfall );
V(ac0) <+ transition (dac0*vdd_t, 0.0, trise, tfall );
V(at1) <+ transition (dat1*vdd_t, 0.0, trise, tfall );
V(ac1) <+ transition (dac1*vdd_t, 0.0, trise, tfall );
V(at2) <+ transition (dat2*vdd_t, 0.0, trise, tfall );
V(ac2) <+ transition (dac2*vdd_t, 0.0, trise, tfall );
V(at3) <+ transition (dat3*vdd_t, 0.0, trise, tfall );
V(ac3) <+ transition (dac3*vdd_t, 0.0, trise, tfall );
V(at4) <+ transition (dat4*vdd_t, 0.0, trise, tfall );
V(ac4) <+ transition (dac4*vdd_t, 0.0, trise, tfall );
V(at5) <+ transition (dat5*vdd_t, 0.0, trise, tfall );
V(ac5) <+ transition (dac5*vdd_t, 0.0, trise, tfall );
V(at6) <+ transition (dat6*vdd_t, 0.0, trise, tfall );
V(ac6) <+ transition (dac6*vdd_t, 0.0, trise, tfall );
V(at7) <+ transition (dat7*vdd_t, 0.0, trise, tfall );
V(ac7) <+ transition (dac7*vdd_t, 0.0, trise, tfall );
V(bt0) <+ transition (dbt0*vdd_t, 0.0, trise, tfall );
V(bc0) <+ transition (dbc0*vdd_t, 0.0, trise, tfall );
V(bt1) <+ transition (dbt1*vdd_t, 0.0, trise, tfall );
V(bc1) <+ transition (dbc1*vdd_t, 0.0, trise, tfall );
V(bt2) <+ transition (dbt2*vdd_t, 0.0, trise, tfall );
```

```
V(bc2) <+ transition (dbc2*vdd_t, 0.0, trise, tfall );
V(bt3) <+ transition (dbt3*vdd_t, 0.0, trise, tfall );
V(bc3) <+ transition (dbc3*vdd_t, 0.0, trise, tfall );
V(bt4) <+ transition (dbt4*vdd_t, 0.0, trise, tfall );
V(bc4) <+ transition (dbc4*vdd_t, 0.0, trise, tfall );
V(bt5) <+ transition (dbt5*vdd_t, 0.0, trise, tfall );
V(bc5) <+ transition (dbc5*vdd_t, 0.0, trise, tfall );
V(bt6) <+ transition (dbt6*vdd_t, 0.0, trise, tfall );
V(bc6) <+ transition (dbc6*vdd_t, 0.0, trise, tfall );
V(bt7) <+ transition (dbt7*vdd_t, 0.0, trise, tfall );
V(bc7) <+ transition (dbc7*vdd_t, 0.0, trise, tfall );
V(flag_sample) <+ transition(cont_flag_sample*vdd_t, 0.0, trise, tfall );

@(final_step)
 $strobe("the total numer of lost clocks =%d",lost_clocks);
         $fclose(fileID);
end
endmodule
```

A.2 Procesador3 para el multiplicador convencional implementado mediante una descripción Verilog-A

```
‘include "discipline.h"
‘include "constants.h"
// very importante because it is determine de behavior
and the measurements of the turtle multiplier
// $Date: 2010/08/24 $
// $Revision: 1.0 $
//
//
// Author: Lancelot Garcia Leyva, Ph.D student
// México: Universidad Autónoma de Tlaxcala
// España: Universitat Politècnica de Catalunya
// transmiter turtle handshake logic
//
// INSTANCE parameters
//    vlogic_high = output voltage for high  [V]
//    vlogic_low   = output voltage for high  [V]
//    vtrans      = voltages above this at input are considered high  [V]
//    tdel, trise, tfall = {usual} [s]
//

module receiver_convetional(clkt, pt0, pt1, pt2, pt3, pt4, pt5, pt6, pt7,
pt8, pt9, pt10, pt11, pt12, pt13, pt14, pt15, flag_sample);
input clkt, flag_sample;

input pt0, pt1, pt2, pt3, pt4, pt5, pt6, pt7,
pt8, pt9, pt10, pt11, pt12, pt13, pt14, pt15;

electrical ackt, reqt, clkt, pt0, pt1, pt2, pt3, pt4, pt5, pt6, pt7,
pt8, pt9, pt10, pt11, pt12, pt13, pt14, pt15, flag_sample;

integer dclkt;
integer dpt0, dpt1, dpt2, dpt3, dpt4, dpt5, dpt6, dpt7;
integer dpt8, dpt9, dpt10, dpt11, dpt12, dpt13,
dpt14, dpt15, cont_flag_sample;

integer cont_clk;
integer fileID_receiver,fileID_receiver1, fileID_receiver2,
```

```
fileID_receiver3, fileID_receiver4, fileID_receiver5,
fileID_receiver6, fileID_receiver7, fileID_receiver8, fileID_receiver9;
    // son las variables para manipular archivos
real var_T;
parameter real trise = 0.1n from (0:inf);
parameter real tfall = 0.1n from (0:inf);
parameter real vdd_t = 0.5; // voltage used for the power supply.
parameter real vlogic_high = vdd_t;
parameter real vlogic_low = 0;
parameter real vtrans = vdd_t/2;


////////////////////////////////////////////////////
//se inicializan las variables ////////////////////////////

   analog begin

@ ( initial_step )
begin
cont_flag_sample=15;
// archivos de excel con tabuladores
fileID_receiver = $fopen("/homespin/users/lgleyva/receiver_conv.csv");
fileID_receiver1 = $fopen("/homespin/users/lgleyva/receiver_conv1.csv");
fileID_receiver2 = $fopen("/homespin/users/lgleyva/receiver_conv2.csv");
fileID_receiver3 = $fopen("/homespin/users/lgleyva/receiver_conv3.csv");
fileID_receiver4 = $fopen("/homespin/users/lgleyva/receiver_conv4.csv");
fileID_receiver5 = $fopen("/homespin/users/lgleyva/receiver_conv5.csv");
fileID_receiver6 = $fopen("/homespin/users/lgleyva/receiver_conv6.csv");
fileID_receiver7 = $fopen("/homespin/users/lgleyva/receiver_conv7.csv");
fileID_receiver8 = $fopen("/homespin/users/lgleyva/receiver_conv8.csv");
fileID_receiver9 = $fopen("/homespin/users/lgleyva/receiver_conv9.csv");
cont_clk=0;
end

////////////////////////////////////////////////////////////////////////
//    se asignan valores de entrada alas variables internas logicackt_c//
dclkt = V(clkt) > vtrans;
dpt0 = V(pt0) > vtrans;
dpt1 = V(pt1) > vtrans;
dpt2 = V(pt2) > vtrans;
dpt3 = V(pt3) > vtrans;
dpt4 = V(pt4) > vtrans;
dpt5 = V(pt5) > vtrans;
dpt6 = V(pt6) > vtrans;
```

```
dpt7 = V(pt7) > vtrans;
dpt8 = V(pt8) > vtrans;
dpt9 = V(pt9) > vtrans;
dpt10 = V(pt10) > vtrans;
dpt11 = V(pt11) > vtrans;
dpt12 = V(pt12) > vtrans;
dpt13 = V(pt13) > vtrans;
dpt14 = V(pt14) > vtrans;
dpt15 = V(pt15) > vtrans;



////////////////////////////////////////////////
// reloj
      @ (cross(V(clkt) - vtrans,  1))  dclkt = 1;
      @ (cross(V(clkt) - vtrans, -1))  dclkt = 0;


////////////////////////////////////////////////////////////////
//  Datos p0t,p0c,.....,p15t,p15c                             ///

      @ (cross(V(pt0) - vtrans,  1))  dpt0 = 1;
      @ (cross(V(pt0) - vtrans, -1))  dpt0 = 0;
      @ (cross(V(pt1) - vtrans,  1))  dpt1 = 1;
      @ (cross(V(pt1) - vtrans, -1))  dpt1 = 0;
      @ (cross(V(pt2) - vtrans,  1))  dpt2 = 1;
      @ (cross(V(pt2) - vtrans, -1))  dpt2 = 0;
      @ (cross(V(pt3) - vtrans,  1))  dpt3 = 1;
      @ (cross(V(pt3) - vtrans, -1))  dpt3 = 0;
      @ (cross(V(pt4) - vtrans,  1))  dpt4 = 1;
      @ (cross(V(pt4) - vtrans, -1))  dpt4 = 0;
      @ (cross(V(pt5) - vtrans,  1))  dpt5 = 1;
      @ (cross(V(pt5) - vtrans, -1))  dpt5 = 0;
      @ (cross(V(pt6) - vtrans,  1))  dpt6 = 1;
      @ (cross(V(pt6) - vtrans, -1))  dpt6 = 0;
      @ (cross(V(pt7) - vtrans,  1))  dpt7 = 1;
      @ (cross(V(pt7) - vtrans, -1))  dpt7 = 0;
      @ (cross(V(pt8) - vtrans,  1))  dpt8 = 1;
      @ (cross(V(pt8) - vtrans, -1))  dpt8 = 0;
      @ (cross(V(pt9) - vtrans,  1))  dpt9 = 1;
      @ (cross(V(pt9) - vtrans, -1))  dpt9 = 0;
      @ (cross(V(pt10) - vtrans,  1))  dpt10 = 1;
      @ (cross(V(pt10) - vtrans, -1))  dpt10 = 0;
      @ (cross(V(pt11) - vtrans,  1))  dpt11 = 1;
```

```
      @ (cross(V(pt11) - vtrans, -1))  dpt11 = 0;
      @ (cross(V(pt12) - vtrans,  1))  dpt12 = 1;
      @ (cross(V(pt12) - vtrans, -1))  dpt12 = 0;
      @ (cross(V(pt13) - vtrans,  1))  dpt13 = 1;
      @ (cross(V(pt13) - vtrans, -1))  dpt13 = 0;
      @ (cross(V(pt14) - vtrans,  1))  dpt14 = 1;
      @ (cross(V(pt14) - vtrans, -1))  dpt14 = 0;
      @ (cross(V(pt15) - vtrans,  1))  dpt15 = 1;
      @ (cross(V(pt15) - vtrans, -1))  dpt15 = 0;


      @ (cross(V(flag_sample) - vtrans, +1))
       begin
       cont_flag_sample=cont_flag_sample+1;
             if (cont_flag_sample==1)
       begin
$fstrobe(fileID_receiver1,"%g,%d,%d,%d,%d,%d,%d,%d,
%d,%d,%d,%d,%d,%d,%d,%d,%d,%d",
$abstime,dpt15,dpt14,dpt13,dpt12,dpt11,dpt10,dpt9,dpt8,
dpt7,dpt6,dpt5,dpt4,dpt3,dpt2,dpt1,dpt0);
end else
if (cont_flag_sample==2)
       begin
$fstrobe(fileID_receiver2,"%g,%d,%d,%d,%d,%d,%d,%d,
%d,%d,%d,%d,%d,%d,%d,%d,%d,%d",
$abstime,dpt15,dpt14,dpt13,dpt12,dpt11,dpt10,dpt9,dpt8,
dpt7,dpt6,dpt5,dpt4,dpt3,dpt2,dpt1,dpt0);
end else
if (cont_flag_sample==3)
       begin
$fstrobe(fileID_receiver3,"%g,%d,%d,%d,%d,%d,%d,%d,
%d,%d,%d,%d,%d,%d,%d,%d,%d,%d",
$abstime,dpt15,dpt14,dpt13,dpt12,dpt11,dpt10,dpt9,dpt8,
dpt7,dpt6,dpt5,dpt4,dpt3,dpt2,dpt1,dpt0);
end else
if (cont_flag_sample==4)
       begin
$fstrobe(fileID_receiver4,"%g,%d,%d,%d,%d,%d,%d,%d,
%d,%d,%d,%d,%d,%d,%d,%d,%d,%d",
$abstime,dpt15,dpt14,dpt13,dpt12,dpt11,dpt10,dpt9,dpt8,
dpt7,dpt6,dpt5,dpt4,dpt3,dpt2,dpt1,dpt0);
end else
if (cont_flag_sample==5)
       begin
```

```
$fstrobe(fileID_receiver5,"%g,%d,%d,%d,%d,%d,%d,%d,
%d,%d,%d,%d,%d,%d,%d,%d,%d",
$abstime,dpt15,dpt14,dpt13,dpt12,dpt11,dpt10,dpt9,dpt8,
dpt7,dpt6,dpt5,dpt4,dpt3,dpt2,dpt1,dpt0);
end else
if (cont_flag_sample==6)
       begin
$fstrobe(fileID_receiver6,"%g,%d,%d,%d,%d,%d,%d,%d,
%d,%d,%d,%d,%d,%d,%d,%d,%d",
$abstime,dpt15,dpt14,dpt13,dpt12,dpt11,dpt10,dpt9,dpt8,
dpt7,dpt6,dpt5,dpt4,dpt3,dpt2,dpt1,dpt0);
end else
if (cont_flag_sample==7)
       begin
$fstrobe(fileID_receiver7,"%g,%d,%d,%d,%d,%d,%d,%d,
%d,%d,%d,%d,%d,%d,%d,%d,%d",
$abstime,dpt15,dpt14,dpt13,dpt12,dpt11,dpt10,dpt9,dpt8,
dpt7,dpt6,dpt5,dpt4,dpt3,dpt2,dpt1,dpt0);
end else
if (cont_flag_sample==8)
       begin
$fstrobe(fileID_receiver8,"%g,%d,%d,%d,%d,%d,%d,%d,
%d,%d,%d,%d,%d,%d,%d,%d,%d",
$abstime,dpt15,dpt14,dpt13,dpt12,dpt11,dpt10,dpt9,dpt8,
dpt7,dpt6,dpt5,dpt4,dpt3,dpt2,dpt1,dpt0);
end else
if (cont_flag_sample==9)
       begin
$fstrobe(fileID_receiver9,"%g,%d,%d,%d,%d,%d,%d,%d,
%d,%d,%d,%d,%d,%d,%d,%d,%d",
$abstime,dpt15,dpt14,dpt13,dpt12,dpt11,dpt10,dpt9,dpt8,
dpt7,dpt6,dpt5,dpt4,dpt3,dpt2,dpt1,dpt0);
end

end

      @ (cross(V(clkt) - vtrans, +1) )
      begin
       cont_clk= cont_clk +1;
       $fstrobe(fileID_receiver,"T%d", cont_clk);
       cont_flag_sample=-1;
      end

@(final_step)
```

```
  $fclose(fileID_receiver);
  $fclose(fileID_receiver1);
  $fclose(fileID_receiver2);
  $fclose(fileID_receiver3);
  $fclose(fileID_receiver4);
  $fclose(fileID_receiver5);
  $fclose(fileID_receiver6);
  $fclose(fileID_receiver7);
  $fclose(fileID_receiver8);
  $fclose(fileID_receiver9);
end
endmodule
```

A.3 Processor3 para el multiplicador Tortuga implementado mediante una descripción Verilog-A

```
// Author: Lancelot Garcia Leyva, Ph.D student
// México: Universidad Autónoma de Tlaxcala
// España: Universitat Politècnica de Catalunya
// receiver turtle handshake logic
// $Date: 2010/08/24 $
// $Revision: 1.0 $

#include "discipline.h"
#include "constants.h"

// INSTANCE parameters
// vlogic_high = output voltage for high  [V]
// vlogic_low   = output voltage for high  [V]
// vtrans      = voltages above this at input are considered high  [V]
// tdel, trise, tfall = {usual} [s]

module receiver_ss(clkt, clkc, reqt, reqc, ackt, ackc, pt0, pc0,
pt1, pc1, pt2, pc2, pt3, pc3, pt4, pc4, pt5, pc5, pt6, pc6, pt7,
pc7, pt8, pc8, pt9, pc9, pt10, pc10, pt11, pc11, pt12, pc12, pt13,
pc13, pt14, pc14, pt15, pc15, flag_sample);
input ackt, ackc;
output reqt, reqc;
input clkt, clkc, flag_sample;

input pt0, pc0, pt1, pc1, pt2, pc2, pt3, pc3, pt4, pc4, pt5,
pc5, pt6, pc6, pt7, pc7, pt8, pc8, pt9, pc9, pt10, pc10, pt11,
pc11, pt12, pc12, pt13, pc13, pt14, pc14, pt15, pc15;

electrical ackt, ackc, reqt, reqc, clkt, clkc, pt0, pc0, pt1,
pc1, pt2, pc2, pt3, pc3, pt4, pc4, pt5, pc5, pt6, pc6, pt7,
pc7, pt8, pc8, pt9, pc9, pt10, pc10, pt11, pc11, pt12, pc12,
pt13, pc13, pt14, pc14, pt15, pc15, flag_sample;

integer dclkt, dclkc;
integer dpt0, dpc0, dpt1, dpc1, dpt2, dpc2, dpt3, dpc3, dpt4,
dpc4, dpt5, dpc5, dpt6, dpc6, dpt7, dpc7;
integer dpt8, dpc8, dpt9, dpc9, dpt10, dpc10, dpt11, dpc11,
dpt12, dpc12, dpt13, dpc13, dpt14, dpc14, dpt15, dpc15,
```

```
cont_flag_sample;

integer logica_ackt, logica_ackc, logica_reqt, logica_reqc;
integer dec_A, dec_B, dec_AB, cont_clk;
// son las variables para manipular archivos
integer fileID_receiver4;
real var_T;
parameter real trise = 0.1n from (0:inf);
parameter real tfall = 0.1n from (0:inf);
parameter real vdd_t = 0.5; // voltage used for the power supply.
parameter real vlogic_high = vdd_t;
parameter real vlogic_low = 0;
parameter real vtrans = vdd_t/2;


////////////////////////////////////////////////////
//se inicializan las variables ////////////////////////////

   analog begin

@ ( initial_step )
begin
logica_reqt=1; logica_reqc=0;  cont_flag_sample=100;
//fileID_receiver = $fopen("/homespin/users/lgleyva/receiver_ss.csv");
//fileID_receiver1 = $fopen("/homespin/users/lgleyva/receiver_ss1.csv");
//fileID_receiver2 = $fopen("/homespin/users/lgleyva/receiver_ss2.csv");
//fileID_receiver3 = $fopen("/homespin/users/lgleyva/receiver_ss3.csv");
// archivo de excel con tabuladores
fileID_receiver4 = $fopen("/homespin/users/lgleyva/receiver_ss4.csv");
//fileID_receiver5 = $fopen("/homespin/users/lgleyva/receiver_ss5.csv");
//fileID_receiver6 = $fopen("/homespin/users/lgleyva/receiver_ss6.csv");
//fileID_receiver7 = $fopen("/homespin/users/lgleyva/receiver_ss7.csv");
//fileID_receiver8 = $fopen("/homespin/users/lgleyva/receiver_ss8.csv");
//fileID_receiver9 = $fopen("/homespin/users/lgleyva/receiver_ss9.csv");
cont_clk=0;
end

////////////////////////////////////////////////////////////////////////
//    se asignan valores de entrada alas variables internas logicackt_c//
logica_ackt = V(ackt) > vtrans;
logica_ackc = V(ackc) > vtrans;
dclkt = V(clkt) > vtrans;
dclkc = V(clkc) > vtrans;
dpt0 = V(pt0) > vtrans;
```

```
dpt1 = V(pt1) > vtrans;
dpt2 = V(pt2) > vtrans;
dpt3 = V(pt3) > vtrans;
dpt4 = V(pt4) > vtrans;
dpt5 = V(pt5) > vtrans;
dpt6 = V(pt6) > vtrans;
dpt7 = V(pt7) > vtrans;
dpt8 = V(pt8) > vtrans;
dpt9 = V(pt9) > vtrans;
dpt10 = V(pt10) > vtrans;
dpt11 = V(pt11) > vtrans;
dpt12 = V(pt12) > vtrans;
dpt13 = V(pt13) > vtrans;
dpt14 = V(pt14) > vtrans;
dpt15 = V(pt15) > vtrans;
dpc0 = V(pc0) > vtrans;
dpc1 = V(pc1) > vtrans;
dpc2 = V(pc2) > vtrans;
dpc3 = V(pc3) > vtrans;
dpc4 = V(pc4) > vtrans;
dpc5 = V(pc5) > vtrans;
dpc6 = V(pc6) > vtrans;
dpc7 = V(pc7) > vtrans;
dpc8 = V(pc8) > vtrans;
dpc9 = V(pc9) > vtrans;
dpc10 = V(pc10) > vtrans;
dpc11 = V(pc11) > vtrans;
dpc12 = V(pc12) > vtrans;
dpc13 = V(pc13) > vtrans;
dpc14 = V(pc14) > vtrans;
dpc15 = V(pc15) > vtrans;


////////////////////////////////////////////////
// reloj
      @ (cross(V(clkt) - vtrans,  1))  dclkt = 1;
      @ (cross(V(clkt) - vtrans, -1))  dclkt = 0;
      @ (cross(V(clkc) - vtrans,  1))  dclkc = 1;
      @ (cross(V(clkc) - vtrans, -1))  dclkc = 0;

////////////////////////////////////////////////
      @ (cross(V(ackt) - vtrans,  1))  logica_ackt = 1;
      @ (cross(V(ackt) - vtrans, -1))  logica_ackt = 0;
      @ (cross(V(ackc) - vtrans,  1))  logica_ackc = 1;
```

```
      @ (cross(V(ackc) - vtrans, -1))  logica_ackc = 0;


////////////////////////////////////////////////////////////////////
//  Datos p0t,p0c,.....,p15t,p15c                                 ///

      @ (cross(V(pt0) - vtrans,  1))  dpt0 = 1;
      @ (cross(V(pt0) - vtrans, -1))  dpt0 = 0;
      @ (cross(V(pc0) - vtrans,  1))  dpc0 = 1;
      @ (cross(V(pc0) - vtrans, -1))  dpc0 = 0;

      @ (cross(V(pt1) - vtrans,  1))  dpt1 = 1;
      @ (cross(V(pt1) - vtrans, -1))  dpt1 = 0;
      @ (cross(V(pc1) - vtrans,  1))  dpc1 = 1;
      @ (cross(V(pc1) - vtrans, -1))  dpc1 = 0;

      @ (cross(V(pt2) - vtrans,  1))  dpt2 = 1;
      @ (cross(V(pt2) - vtrans, -1))  dpt2 = 0;
      @ (cross(V(pc2) - vtrans,  1))  dpc2 = 1;
      @ (cross(V(pc2) - vtrans, -1))  dpc2 = 0;

      @ (cross(V(pt3) - vtrans,  1))  dpt3 = 1;
      @ (cross(V(pt3) - vtrans, -1))  dpt3 = 0;
      @ (cross(V(pc3) - vtrans,  1))  dpc3 = 1;
      @ (cross(V(pc3) - vtrans, -1))  dpc3 = 0;

      @ (cross(V(pt4) - vtrans,  1))  dpt4 = 1;
      @ (cross(V(pt4) - vtrans, -1))  dpt4 = 0;
      @ (cross(V(pc4) - vtrans,  1))  dpc4 = 1;
      @ (cross(V(pc4) - vtrans, -1))  dpc4 = 0;

      @ (cross(V(pt5) - vtrans,  1))  dpt5 = 1;
      @ (cross(V(pt5) - vtrans, -1))  dpt5 = 0;
      @ (cross(V(pc5) - vtrans,  1))  dpc5 = 1;
      @ (cross(V(pc5) - vtrans, -1))  dpc5 = 0;

      @ (cross(V(pt6) - vtrans,  1))  dpt6 = 1;
      @ (cross(V(pt6) - vtrans, -1))  dpt6 = 0;
      @ (cross(V(pc6) - vtrans,  1))  dpc6 = 1;
      @ (cross(V(pc6) - vtrans, -1))  dpc6 = 0;

      @ (cross(V(pt7) - vtrans,  1))  dpt7 = 1;
      @ (cross(V(pt7) - vtrans, -1))  dpt7 = 0;
      @ (cross(V(pc7) - vtrans,  1))  dpc7 = 1;
```

```
@ (cross(V(pc7) - vtrans, -1))  dpc7 = 0;

@ (cross(V(pt8) - vtrans,  1))  dpt8 = 1;
@ (cross(V(pt8) - vtrans, -1))  dpt8 = 0;
@ (cross(V(pc8) - vtrans,  1))  dpc8 = 1;
@ (cross(V(pc8) - vtrans, -1))  dpc8 = 0;

@ (cross(V(pt9) - vtrans,  1))  dpt9 = 1;
@ (cross(V(pt9) - vtrans, -1))  dpt9 = 0;
@ (cross(V(pc9) - vtrans,  1))  dpc9 = 1;
@ (cross(V(pc9) - vtrans, -1))  dpc9 = 0;

@ (cross(V(pt10) - vtrans,  1))  dpt10 = 1;
@ (cross(V(pt10) - vtrans, -1))  dpt10 = 0;
@ (cross(V(pc10) - vtrans,  1))  dpc10 = 1;
@ (cross(V(pc10) - vtrans, -1))  dpc10 = 0;

@ (cross(V(pt11) - vtrans,  1))  dpt11 = 1;
@ (cross(V(pt11) - vtrans, -1))  dpt11 = 0;
@ (cross(V(pc11) - vtrans,  1))  dpc11 = 1;
@ (cross(V(pc11) - vtrans, -1))  dpc11 = 0;

@ (cross(V(pt12) - vtrans,  1))  dpt12 = 1;
@ (cross(V(pt12) - vtrans, -1))  dpt12 = 0;
@ (cross(V(pc12) - vtrans,  1))  dpc12 = 1;
@ (cross(V(pc12) - vtrans, -1))  dpc12 = 0;

@ (cross(V(pt13) - vtrans,  1))  dpt13 = 1;
@ (cross(V(pt13) - vtrans, -1))  dpt13 = 0;
@ (cross(V(pc13) - vtrans,  1))  dpc13 = 1;
@ (cross(V(pc13) - vtrans, -1))  dpc13 = 0;

@ (cross(V(pt14) - vtrans,  1))  dpt14 = 1;
@ (cross(V(pt14) - vtrans, -1))  dpt14 = 0;
@ (cross(V(pc14) - vtrans,  1))  dpc14 = 1;
@ (cross(V(pc14) - vtrans, -1))  dpc14 = 0;

@ (cross(V(pt15) - vtrans,  1))  dpt15 = 1;
@ (cross(V(pt15) - vtrans, -1))  dpt15 = 0;
@ (cross(V(pc15) - vtrans,  1))  dpc15 = 1;
@ (cross(V(pc15) - vtrans, -1))  dpc15 = 0;

@ (cross(V(flag_sample) - vtrans, +1))
 begin
```

```
       cont_flag_sample=cont_flag_sample+1;
             if (cont_flag_sample==40)
       begin
$fstrobe(fileID_receiver4,"%g,%d,%d,%d,%d,%d,%d,%d,%d,
%d,%d,%d,%d,%d,%d,%d,%d",$abstime,dpt15,dpt14,dpt13,
dpt12,dpt11,dpt10,dpt9,dpt8,dpt7,dpt6,dpt5,dpt4,dpt3,
dpt2,dpt1,dpt0);
end
end


      @ (cross(V(clkt) - vtrans, +1) or cross(V(clkc) - vtrans, -1))
      begin
       if (dclkt != dclkc)
       begin
       cont_clk= cont_clk +1;
       //$fstrobe(fileID_receiver,"T%d", cont_clk);
       cont_flag_sample=-1;
       end else begin
       //$fstrobe(fileID_receiver,"Lost Clock");
       cont_flag_sample=-1;
       end
      end


//if ((logica_reqt != logica_reqc) && logica_reqt)begin end
V(reqt) <+ transition (logica_reqt*vdd_t, 0.0, trise, tfall );
V(reqc) <+ transition (logica_reqc*vdd_t, 0.0, trise, tfall );


@(final_step)
  //$fclose(fileID_receiver);
  //$fclose(fileID_receiver1);
  //$fclose(fileID_receiver2);
  //$fclose(fileID_receiver3);
  $fclose(fileID_receiver4);
  //$fclose(fileID_receiver5);
  //$fclose(fileID_receiver6);
  //$fclose(fileID_receiver7);
  //$fclose(fileID_receiver8);
  //$fclose(fileID_receiver9);
end

endmodule
```

Lista de publicaciones

1. Garcia-Leyva, L.; Calomarde, A.; Moll, F.; Rubio, A.; "New redundant logic function design method for extremely high noise and low-voltage scenarios", XXIV Conference on Design of Circuits and Integrated Systems. November 18-20,2009, Zaragoza, Spain.

2. Garcia-Leyva, L.; Calomarde, A.; Moll, F.; Rubio, A.; "Turtle Logic: A new probabilistic design methodology of nanoscale digital circuits", Circuits and Systems (MWSCAS), 2010 53rd IEEE International Midwest Symposium on. Publication Year: 2010 , Page(s): 1101 - 1104. ISBN: 9781424477722. ISSN : 1548-3746.

3. Garcia-Leyva, L.; Calomarde, A.; Moll, F.; Rubio, A.; "New Methodology of Digital IC Design in extremely high noise and low voltage Scenarios", Proceedings of the 1st Barcelona Forum on Ph.D. Research in Electronic Engineering, ISBN: 978-84-7653- 398-7.

4. Garcia-Leyva, L.; Calomarde, A.; Moll, F.; Rubio, A.; "Turtle logic: Novel IC digital probabilistic design methodology", Proceedings of the Barcelona Forum on Ph.D. Research in Communications, Electronics and Signal Processing. Barcelona, October 2010. ISBN: 978-84-7653-495-3.

5. Garcia-Leyva, L.; Calomarde, A.; Moll, F.; Rubio, A.; “A new probabilistic design methodology of nanoscale digital circuits”, 21st International Conference on Electronics Communications and Computers CONIELECOMP 2011, February 28th - March 2nd, 2011. ISBN: 978-1-4244-9557-3.

6. Garcia-Leyva, L.; Andrade, D.; Gómez, S.; Calomarde, A.; Moll, F.; Rubio, A.; ‘New redundant logic design concept for high noise and low voltage scenarios”, Microelectronics journal, Elsevier. Volume 42, Issue 12, December 2011, Pages 1359–1369.

7. Garcia-Leyva, L.; Calomarde, A.; Moll, F.; Rubio, A.; “Novel redundant logic design for noisy low voltage scenarios ”, 4th IEEE LASCAS - Latin American Symposium on Circuits and Systems – LASCAS 2013 - CUZCO, PERU from 27th February to 1st March, 2013.

Patente

1. Patente Europea y PCT número P200901626 con título:‘Procedimiento para la mejora de la fiabilidad de circuitos Integrados Digitales en condiciones de baja relación señal a ruido”, autores: Calomarde, A.; Moll, F.; Rubio, A. y Garcia-Leyva, L.

Bibliografía

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics Magazine*, pp. 114–117, Apr. 1965.

[2] L. Serge, X. Jimmy, and Z. Alex, "Future trends in microelectronics: The Nano, the Giga, and the Ultra," *ISBN: 0471484059, New york*, 2004.

[3] A. Bachtold, P. Hadley, T. Nakanishi, and C. Dekker, "Logic circuits based on carbon nanotubes," *Physica E*, vol. 16, pp. 42–46, 2003.

[4] K. K. Likharev, "Single-electron devices and their applications," *Proceedings of the IEEE*, vol. 87, pp. 606–632, Apr. 1999.

[5] I. Zutic, J. Fabian, and S. D. Sarma, "Spintronics: Fundamentals and applications," *Reviews of Modern Physics*, vol. 76, pp. 323–410, Apr. 2004.

[6] ITRS, "http://www.itrs.net/," *International Technology Roadmap for Semiconductors*, 2011.

[7] A. V. Mezhiba and E. G. Friedman, "Scaling trends of on-chip power distribution noise," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 12, pp. 386–394, Apr. 2004.

[8] N. Sano, "Increasing importance of electronic thermal noise in sub-0.1mm Si-MOSFETs," *IEICE Transactions on Electronics*, vol. E83-C, pp. 1203–1211, Aug. 2000.

[9] M. Van Heijningen, M. Badaroglu, S. Donnay, G. G. E. Gielen, and H. J. D. Man, "Substrate noise generation in complex digital systems: Efficient modeling and simulation methodology and experimental verification," *IEEE J. of Solid-State Circuits*, vol. 37, pp. 1065–1072, Aug. 2002.

[10] M. A. Elgamel and M. A. Bayoumi, "Interconnect noise analysis and optimization in deep submicron technology," *IEEE Circuits and Systems Magazine*, vol. 3, pp. 6–17, 2003.

[11] O. A. Amusan, A. F. Witulski, L. W. Massengill, B. L. Bhuva, P. R. Fleming, M. L. Alles, A. L. Sternberg, J. D. Black, and R. D. Schrimpf, "Charge collection and charge sharing in an 130 nm CMOS technology," *IEEE Tran. Nuclear Science*, vol. 53, pp. 3253–3258, Dec. 2006.

[12] J. C. Ebergen, J. Segers, and I. Benko, *Parallel Program and Asynchronous Circuit Design.* Computer Science Department, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1, Revised March 30, 1994.

[13] C. M. Grinstead and J. L. Snell, "Central limit theorem, introduction to probability," *AMS Bookstore, ISBN 0821807498. 1997*, vol. Chapter 9. Second edition, pp. 325–360, 1997.

[14] A. Sunami Hideo, *Dimension Increase in Metal-Oxide-Semiconductor Memories and Transistors.* Advances in Solid Circuit Technologies, 2010.

[15] ——, "Parallel progress and asynchronous circuit design," *in Asynchronous Digital Circuit Design, G. Birtwistle and A. Davis eds. New Work: Springer-Verlag*, pp. 51–103, 1995.

[16] M. Omana, D. Rossi, and C. Metra, "Novel transient fault hardened static latch," *Proceedings of the International Test Conference, ITC 2003*, pp. 886–892, Sept.30-Oct.2, 2003.

[17] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. Kim, "Robust system design with built-in soft error resilience," *IEEE Computer*, vol. 38, no. 2, pp. 43–52, Feb. 2005.

[18] A. Goel, S. Bhunia, H. Mahmoodi, and K. Roy, "Low-overhead design of soft-error-tolerant scan flip-flops with enhanced-scan capability," *Proccedding of Asia and South Pacific Conference on Design Automation*, p. 6, 24-27 Jan. 2006.

[19] A. Drake, A. KleinOsowski, and A. Martin, "A self-correcting soft error tolerant flop-flop," *12th NASA Symposium on VLSI Design, Coeur d'Alene, Idaho, USA*, Oct. 4-5, 2005.

[20] O. Roystein, J. Aditya, and J. C. Tapan, "A tmr scheme for seu mitigation in scan flip-flops," *Proceedings of the 8th International Symposium on Quality Electronic Design (ISQED'07), IEEE Computer Society*, 2007.

[21] F. Mendoza-Hernandez, M. Linares-Aranda, and V. H. Champac-Vilela, "The noise immunity of dynamic digital circuits with technology scaling," *in Proc. of Int. Symp. Circuits and Systems ISCAS*, pp. 493–496, May 2004.

[22] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, K. A., and D. V., "Parameter variations and impact on circuits and microarchitecture," *in Proc. Design Automation Conf. DAC*, pp. 338–342, June 2003.

[23] S. Bhunia, S. Mukhopadhyay, and R. K., "Process variations and process-tolerant design," *in Proc. Int. Conf. VLSI Design*, pp. 699–704, Jan. 2007.

[24] J. V. Neumann, *Probabilistic Logics the Synthesis of Reliable Organisms from Unreliable Components Automata Studies*, C. E. Shannon and J. McCarthy, Eds. C. E. Shannon and J. McCarthy, 1956.

[25] S. Spagocci and T. Fountain, "Fault rates in nanochip devices," *Proc. Electrochemical Society, The Electrochemical society INc.*, vol. 98-99, pp. 582–593, 1999.

[26] P. Korkmaz, "Probabilistic CMOS PCMOS in the nanoelectronics regime," Ph.D. dissertation, Georgia Institute of Technology, Dec. 2007.

[27] R. Iris Bahar, J. Mundy, and J. Chen, "A probabilistic-based design methodology for nanoscale computation," *Computer Aided Design IC-CAD*, vol. ISBN: 1-58113-762-1, pp. 480– 486, International Conference on Publication Date: 9-13 Nov. 2003.

[28] D. E. Muller and W. S. Bartky, "A theory of asynchronous circuits," *In Proceedings of an International Symposium on the Theory of Switching*, pp. 204–243, Apr. 1959.

[29] W. A. Clark, "Macromodular computer systems," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, ser. AFIPS '67 Spring. New York, NY, USA: ACM, 1967, pp. 335–336. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465536

[30] W. Qian, D. R. Marc, Z. Hongchao, and B. Jehoshua, "Transforming probabilities with combinational logic," *Journal IEEE Transactions on Computer-Aided design in Integrated Circuits and Systems*, vol. 30, no. 9, pp. 1279–1292, September 2011.

[31] I. E. Sutherland, "Micropipelines," *Communications of ACM*, vol. 32, pp. 720–738, June 1989.

[32] A. J. Martin, "Formal progress transformations for VLSI circuit synthesis," *in Formal Development of Programs and Proofs E. W. Dijkstra, Ed. reading, MA: Addison-wesley*, pp. 59–80, 1989.

[33] K. van Berkel, "Beware the isochronic fork," *Integr. VLSI J.*, vol. 13, no. 2, pp. 103–128, June 1992. [Online]. Available: http://dx.doi.org/10.1016/0167-9260(92)90001-F

[34] O. Elissati, E. Yahya, S. Rieubon, and L. Fesquet, "Optimizing and comparing CMOS implementations of the C-element in 65nm technology: self-timed ring case," *Proceedings of the 20th international conference on Integrated circuit and system design: power and timing modeling, optimization and simulation*, pp. 137–149, 2011.

[35] M. Shams, "Optimizing CMOS implementations of the C-element," *Computer Design: VLSI in Computers and Processors, 1997. ICCD '97. Proceedings., 1997 IEEE International Conference on*, pp. 700 – 705, 1997.

[36] S. Furber, "Computing without clocks: Micropipelining the ARM processor." *In G. Birtwistle and A. Davis, editor, Asinchronous Digital Circuit Design, Workshop in computing, Springer-Verlag*, pp. 211–262, 1995.

[37] M. Shams, J. Ebergen, and M. Elmasry, "A comparison of CMOS implementations of an asynchronous circuits primitive: the C-element." *In international Symposium on Low Power Electronics and design*, pp. 93–96, Aug. 1996.

[38] T. Karnik, S. Borkar, and V. De, "Sub-90nm technologies: challenges and opportunities for cad," in *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, ser. ICCAD '02. New York, NY, USA: ACM, 2002, pp. 203–206. [Online]. Available: http://doi.acm.org/10.1145/774572.774602

[39] S. Borkar, T. Karnik, and V. De, "Design and reliability challenges in nanometer technologies," in *Proceedings of the 41st annual Design Automation Conference*, ser. DAC '04. New York, NY, USA: ACM, 2004, pp. 75–75. [Online]. Available: http://doi.acm.org/10.1145/996566.996588

[40] K. P. Parker and M. E. J., "Probabilistic treatment of general combinational networks," *IEEE Transactions on Computers*, vol. 24, no. 6, pp. 668–670, 1975.

[41] J. Savir, G. Dilow, and P. H. Bardell, "Random pattern testability," *IEEE Transactions on Computers*, vol. 33, pp. 79–90, 1984.

[42] J. J. Liou, K. T. Cheng, and A. Krstic, "Fast statistical timing analysis by probabilistic event propagation," *in Design Automation Conference*, pp. 661–666, 2001.

[43] R. Marculescu, D. Marculescu, and M. Pedram, "Logic level power estimation consideringspatiotemporal correlations," *in International Conference on Computer-Aided Design*, pp. 294–299, 1994.

[44] A. Gill, "Synthesis of probability transformers," *Journal of the Franklin Institute*, vol. 274, no. 1, pp. 1–19, 1962.

[45] ——, "On a weight distribution problem, with application to the design of sthocastic generators," *Journal of the ACM*, vol. 10, no. 1, pp. 110–121, 1963.

[46] P. Jeavons, A. Cohen, and J. Shawe-Taylor, "Generating binary sequences for sthocastic computing," *IEEE Transactions on Information Theory*, vol. 40, no. 3, pp. 717–720, 1994.

[47] L. Chakrapani, P. Korkmaz, B. Akgul, and K. Palem, "Probabilities system-on-a-chip architecture," *ACM Transactions on design Automation of Electronic Systems*, vol. 12, no. 3, pp. 1–28, 2007.

[48] D. Wilhelm and J. Bruck, "Sthocastic switching circuit synthesis," *in International Symposium on Information Theory*, pp. 1388–1392, 2008.

[49] C. E. Shanon, "The synthesis of two terminal switching circuits," *Bell System technical Journal*, vol. 28, pp. 59–98, 1949.

[50] H. Zhou and J. Bruck, "On the expressibility of sthocastic switching circuits," *in International symposium of Information Theory*, pp. 2061–2065, 2009.

[51] S. Geman and K. Kochanek, "Dynamic programming and the graphical representation of error-correcting codes," *Information Theory, IEEE Transactions on*, vol. 47 Num. 2, pp. 549–568, February 2001.

[52] C. Constantinescu, "Impact of deep submicron technology on dependability of vlsi circuits," *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, vol. 0, p. 205, 2002.

[53] M. Van Heijningen, M. Badaroglu, S. Donnay, G. Gielen, and H. Man, "Substrate noise generation in complex digital systems: Efficient modeling and simulation methodology and experimental verification," in *IEEE Journal of Solid-State Circuits*, vol. 37, Aug. 2002, pp. 1065–1072.

[54] J. B. Johnson, "Thermal agitation of electrical charge in conductors," *Physical review*, vol. 32, pp. 97–109, July 1928.

[55] A. Van Der Ziel, *Noise: Sources, Characterization, Measurement.* Prentice Hall, 1970.

[56] B. Razavi, *Design of analog CMOS Integrated Circuits.* McGraw-Hill, 2000.

[57] F. A. Bower, D. J. Sorin, and S. Ozev, "A mechanism for online diagnosis of hard faults in microprocessors," *Microarchitecture, 2005. MICRO-38. Proceedings. 38th Annual IEEE-ACM International Symposium on*, Barcelona, Spain, 12-16 November, 2005.

[58] A. Amendola and et. al, "Fault behavior observation of a microprocessor system through a VHDL simulation-based fault injection experiment," *Design Automation Conference, 1996, with EURO-VHDL '96 and Exhibition, Proceedings EURO-DAC '96, European*, pp. 536–541, 16-20 Sep 1996.

[59] H. Zarandi, S. Miremadi, and A. Ejlali, "Fault injection into verilog models for dependability evaluation of digital systems," *Parallel and Distributed Computing, 2003. Proceedings. Second International Symposium on*, pp. 281–287, 13-14 Oct. 2003.

[60] A. Todd, B. David, M. Trevor, and F. Krisztián, "Making typical silicon matter with razor," *IEEE Computer Society*, pp. 57–65, March 2004.

[61] E. Dan, D. Shidhartha, S. Lee, A. David Blaauw, Todd, M. Trevor, S. K. Nam, and F. Krisztián, "Razor: Circuit-level correction of timing errors for low-power operation," *IEEE Computer Society*, pp. 10–20, November-December 2004.

[62] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proceedings of the 36th annual IEEE-ACM International Symposium on Microarchitecture*, ser. MICRO 36. Washington, DC, USA: IEEE Computer Society, December 2003, pp. 7–. [Online]. Available: http://dl.acm.org/citation.cfm?id=956417.956571

[63] K. Nepal, R. Bahar, J. Mundy, W. Patterson, and A. Zaslavsky, "Designing logic circuits for probabilistic computation in the presence of noise," in *Design Automation Conference, 2005. Proceedings. 42nd.*, June 2005, pp. 485–490.

[64] L. Garcia-Leyva, A. Calomarde, F. Moll, and A. Rubio, "Turtle logic: A new probabilistic design methodology of nanoscale digital circuits," in *Circuits and Systems MWSCAS, 2010 53rd IEEE International Midwest Symposium on*, aug. 2010, pp. 1101 –1104.

[65] F. Moll and A. Rubio, "Spurious signals in digital CMOS VLSI circuits: a propagation analysis," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 39, no. 10, pp. 749–752, Oct 1992.

[66] synopsys, "Using noise models," *Star-Hspice Manual*, vol. Release 2001.2, pp. 1–3, June 2001.

[67] ——, "Hspice mosfet models manual," *Star-Hspice Manual*, vol. Version X-2005.09, pp. 206–211, September 2005.

[68] C. Chih-Hung, M. J. Deen, M. Matloubian, and Y. Cheng, "Extraction of the induced gate noise, channel thermal noise and their correlation in sub-micron mosfets from rf noise measurement," *Proc. IEEE 2001 Int. Conference on Microelectronic Test Structures*, vol. 14, pp. 131–135, March 2001.

[69] L. B. Kish, "End of moore's law: thermal (noise) death of integration in micro and nano electronics," *Physics Letters A*, vol. 305, p. 144–149, 2002.

[70] S. Kullback and Leibler, *Information theory and statistics.* New York: Dover, 1969.

[71] M. Morris Mano, *Digital Logic and Computer Design.* Prentice-Hall, 1979.

[72] T. Karnik, P. Hazucha, and J. Patel, "Characterization of soft errors caused by single event upsets in cmos processes," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 2, April-June 2004.

[73] P. E. Dodd, "Physics-based simulation of single-event effects," *IEEE Trans. Device Mater Reliability*, vol. 5, no. 3, pp. 343–357, 2005.

[74] R. Baumman, "Soft errors in advanced computer systems," *IEEE Design Test Computers*, vol. 22, no. 3, pp. 258–266, 2005.

[75] R. R. Rao, D. Blaauw, and D. Sylvester, "Soft error reduction in combinational logic using resizing and flipflop selection," *Proceding IEEE/ACM ICCAD*, pp. 502–509, 2006.

[76] C. Zhao and S. Dey, "Improving transient error tolerance of digital vlsi circuits using robustness compiler (roco)," *in Proceding 7Th International symposium Quality Electronic Design, ISQED*, pp. 502–509, 2006.

[77] U. Diril, S. Dhillon, and A. Chatterjee, "Design of adaptive nanometer digital systems for effective control of soft error tolerance," *in IEEE VLSI Test Symposium VTS*, pp. 298–303, 2005.

[78] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," *International Conference on Dependable Systems and Networks*, pp. 389–398, 2002.

[79] P. Hazucha and et al, "Measurements and analysis of ser-tolerant latch in a 90-nm dual-vt cmos process," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 9, September 2004.

[80] R. Ramanarayanan, V. Degalahal, N. Vijaykrishnan, M. Irwin, and D. Duarte, "Analysis of soft error rate in flip-flops and scannable latches," *IEEE SOC Conference*, pp. 231–234, September 2003.

[81] Q. Zhaou and K. Mohanram, "Cost-effective radiation hardening technique for combinational logic," *ICCAD*, pp. 100–106, 2004.

[82] Y. Arima, T. Yamashita, Y. Komatsu, T. Fujimoto, and K. Ishibashi, "Cosmic-ray immune latch circuit for 90nm technology and beyond,"

Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC. 2004 IEEE International, vol. 1, pp. 492–493, 15-19 Feb. 2004.

[83] L. Rockett, "An SEU hardened CMOS data latch design," *IEEE Transactions on Nuclear Science*, vol. NS-35, no. 6, pp. 1682–1687, Dec. 1988.

[84] S. Whitaker, J. Canaris, and K. Liu, "SEU hardened memory cells for a CCSDS reed-solomon encoder," *IEEE Transactions on Nuclear Science*, vol. 38, no. 6, pp. 1471 – 1477, Part 1, Dec. 1991.

[85] M. Liu and S. Whitaker, "Low power SEU immune CMOS memory circuits," *IEEE Transactions on Nuclear Science*, vol. 39, no. 6, pp. 1679 – 1684, Part 1-2, Dec. 1992.

[86] T. Calin, M. Nicolaidis, and R. Velazco, "Upset hardened memory design for submicron CMOS technology," *IEEE Transactions on Nuclear Science*, vol. 43, no. 6, pp. 2874 – 2878, Part 1, Dec. 1996.

[87] R. Naseer and J. Draper, "The DF-dice storage element for immunity to soft errors," *48th Midwest Symposium on Circuits and Systems*, pp. 303 – 306, 7-10 Aug. 2005.

[88] J. Gambles, L. Miles, J. Hass, W. Smith, S. Whitaker, and B. Smith, "An ultra-low-power, radiation-tolerant reed solomon encoder for space applications," *IEEE Custom Integrated Circuits Conference*, pp. 631 – 634, 21-24 Sept. 2003.

[89] M. Nicolaidis, "Time redundancy based soft-error tolerance to rescue nanometer technologies," *Proceedings VLSI Test Symposium*, pp. 86–94, 1999.

[90] M. D. E. P., "Soft error rate mitigation techniques for modern microcircuits," *International Reliability Physics Symposium*, 2002.

[91] J. B. Cho, J. R. Marum, and G. W. McIver, "Triple redundant fault-tolerant register," *United States Patent 5,031,180*, Jul 9, 1991.

[92] S. Krishnamohan and N. Mahapatra, "A highly-efficient technique for reducing soft errors in static CMOS circuits," *Proc. ACM International Conf. Computer Design (ICCD)*, pp. 126 – 131, Oct. 2004.

[93] ——, "Combining error masking and error detection plus recovery to combat soft errors in static CMOS circuits," *Proc. International Conference on Dependable Systems and Networks*, pp. 40 – 49, July 2005.

[94] P. Elakkumanan, K. Prasad, and R. Sridhar, "Time redundancy based scan flip-flop reuse to reduce SER of combinational logic," *International Symposium on Quality Electronic Design*, 2006.

[95] J. Sparso and S. Furber, *Principles of Asynchronous Circuit Design: A systems Perspective*. Springer, 2001.

[96] B. C.R. and W. B.A., "A two's complement parallel array multiplication algorithm," *Computers, IEEE Transactions on*, vol. C-22, no. 12, pp. 1045 – 1047, dec. 1973.

[97] J. McCanny and J. McWhirter, "Completely iterative, pipelined multiplier array suitable for VLSI," *Electronic Circuits and Systems, IEE Proceedings G*, vol. 129, no. 2, pp. 40 –46, april 1982.

New redundant logic function design method for extremely high noise and low-voltage scenarios

Lancelot García[#*], Antonio Calomarde[*], Francesc Moll[*], Antonio Rubio[*]

[#]*FCBIyT, Tlaxcala Autonomous University*
C/Apizaquito s/n, 90300 Apizaco Tlaxcala, México
[1]lancelot@eel.upc.edu
[*]*Universitat Politècnica de Catalunya*
Dept. of Electronic Engineering
C/Jordi Girona, 31 08034, Barcelona, España
[2]calomarde@eel.upc.edu, [3]moll@eel.upc.edu, [4]antonio.rubio@upc.edu

***Abstract*— The continuing trends of device scaling and increase in complexity towards terascale system on chip level of integration are putting growing difficulties into several areas of design. The intrinsic variability problem is aggravated by variations caused by the difficulties of controlling Critical Dimension (CD) in nanometer technologies. The effect of variability is the difficulty in predicting and designing circuits with precise device and circuit characteristics. In this paper, a new logic gate implementation methodology oriented to special applications or technologies is presented, to improve the gate tolerance to errors due to noise, defects or manufacturability errors. The methodology is based on reinforcement of valid input-output combinations by means of logic feedback functions. Simulations show an excellent performance of our approach in the presence of large random noise at the inputs.**

***Keywords*— Probabilistic, redundancy, low-voltage, high noise, better than worst case.**

I. INTRODUCTION

The continuous scaling down of CMOS transistors following Moore's Law has enabled the design of ever complex electronic products and this improvement has had a great impact on society well fare.

However, as dimensions in devices approach physical limits of semiconductor operation the devices are more susceptible to several problems like properties variability, susceptibility to noise, random behavior and defects. Therefore, devices in ultimately scaled technologies will be increasingly unreliable. This unreliability is expected to even increase as new technologies replace today's CMOS circuits (molectronics, quantum devices, etc.) [1].

Moreover, it is clear that designers will aggressively scale down supply voltage to reduce dynamic power dissipation. The resulting reduction of logic levels approaching the thermal noise limit, and consequently reduced signal to noise margins will expose computation to higher soft-error rates. Therefore, circuit designers can no longer assume that future circuits will have error-free operation.

In order to continue to build reliable circuits with these unreliable components, new design techniques have to be introduced. The problem of designing reliable systems with unreliable components traces back to Von Neumann, who proposed several techniques based on N-tuple Modular Redundancy (NMR) [1] where each gate or block is replicated N times and the final output is obtained from a majority voting gate. Other early approaches to the problem based in NMR include Interwoven Redundant Logic [2], or Quadded Logic [3]. The modules can be single gates, logic blocks, or functional units, depending on the amount of error tolerance the system requires. For the usual case of Triple redundancy (TMR), the system provides relief from an error caused in a single unit, but an error in two or more of the three modules will cause the logic to fail. These methods increase system reliability in the presence of transient faults. Nevertheless, the main assumption underlying these redundancy processes is that the final majority voting gate is perfect and free of faults.

Recently, as the reliability problems associated to technology scaling have become apparent, new proposals have appeared in the literature addressing the problem from the point of view of noise tolerance instead of exclusively considering hard defects. Among the new proposals, there is the so called Probabilistic Logic based on Markov Random Field theory (MRF CMOS) [5] which identifies valid and invalid states, and derives a logic implementation trying to reinforce valid states in order to increase their probability and decrease the probability of erroneous states. Our proposal is based on the same concept of valid and invalid states that the MRF method introduces, but we present a new implementation method that improves the robustness of the logic circuits.

The structure of the paper is as follows: in Section II, the basic concepts and principles of Probabilistic Logic are introduced, as well as the MRF proposal. In Section III, our proposal is introduced, explaining the implementation of a NOT and a NAND gate as examples. Section IV presents simulation results of the different approaches using a 90nm CMOS process technology. Finally, the conclusions are presented in Section V.

II. MRF Probabilistic Logic and Reinforcement

Markov Random Field (MRF) theory is a branch of probability theory widely used in several applications dealing with spatially related random variables [4]. This formulation is applicable in the context of circuit nodes subject to noise and defects, acting as random variables. MRF defines a set of random variables called sites, $X=\{x_1,x_2,...,x_k\}$, where each variable x_i can take on various values called labels. These sites are not independent, but have a statistical dependence between one another, as shown in Figure 1. The probability of a particular site in the neighbourhood depends only on its immediate neighbours, and it is connected by an edge. The edge of the neighbourhood represents the conditional dependence between connected variables in the neighbourhood.

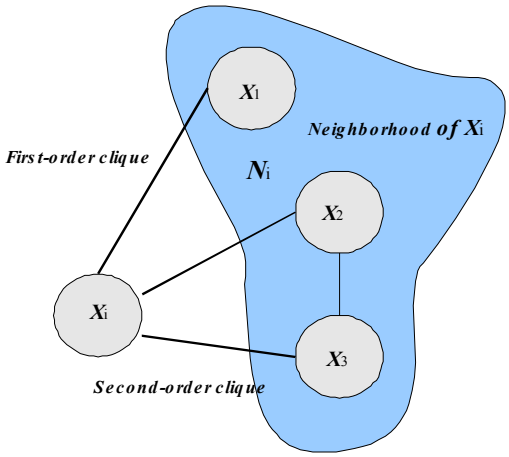


Figure 1. The MRF neighborhood system [5].

In [5], the MRF concepts and formulation were applied to logic circuit design. These definitions can be applied to logic circuits by identifying the nodes of the circuits with the sites. In this application scenario, the edges are determined by the logic gates, which relate outputs to inputs, and the values that the sites can take are either '0' or '1'. Each possible combination of values of the nodes defines a state of the circuit. Note that this definition of state applies to combinational circuits and should not be confused with the conventional definition of state in sequential circuits.

As an example, consider the NOT gate represented in Figure 2. It has two nodes, x0 and x1, which in principle can take any value. Therefore, there are 4 possible states, as shown in Table I. Valid states are those with f=1, and invalid states are those with f=0. The goal is to find a more complex implementation that reinforces the valid states and therefore decreases the probability of the invalid states, making the system more robust to faults. This goal can be achieved by implementing a feedback function taking into account the valid states as indicated in Table I, obtaining an implementation as shown in Figure 3 [5].

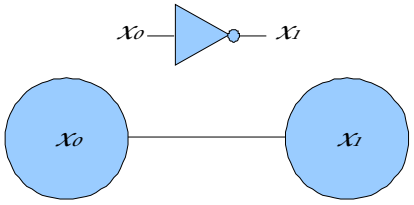


Figure 2. A logic inverter circuit (a) and its dependence graph (b) for Markov Random Field method

This particular implementation and feedback method greatly increases system reliability, but it has the disadvantage that the evolution of states are not taken into account because the implementation is based on a different reasoning. For instance, a fault may induce an invalid state, and the result of that state, due to reinforcement implementation, may be an incorrect state, or an oscillation. An example of such a behavior is a transient fault inducing the state 00. The result of this state is the state 11, and the result of the state 11 is again 00, etc.

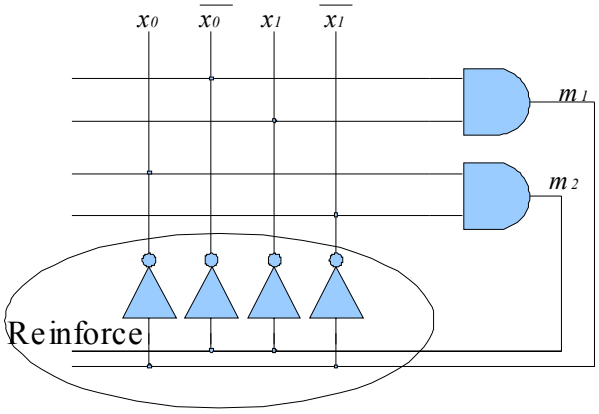


Figure 3. MRF circuit encoding the clique function of two logic variables defining an inverter.

TABLE I
INVERTER LOGIC COMPATIBILITY FUNCTION WITH ALL POSSIBLE CASES, RELATIONSHIPS TO DESIGN AND PROBABILITY FOR ALL STATES RESPECT TO MRF.

x_0	x_1	f	Dec	State	Minterms	reinforce feedback
0	0	0	0	I	$m_0 = \overline{x_0} \cdot \overline{x_1}$	
0	1	1	1	V	$m_1 = \overline{x_0} \cdot x_1$	$x_0 = \overline{m_1}$ $\overline{x_1} = \overline{m_1}$
1	0	1	2	V	$m_2 = x_0 \cdot \overline{x_1}$	$\overline{x_0} = \overline{m_2}$ $x_1 = \overline{m_2}$
1	1	0	3	I	$m_3 = x_0 \cdot x_1$	

I: invalid state; V: valid state

III. NEW PROPOSAL

In this work, we take from [5] the idea of reinforcement taking into account the complete set of possible states. As shown in Figure 3, the resulting implementation of a gate needs both input and output nodes and their complements. However, the MRF implementation implicitly assumes that the complements are correct, and therefore, not all the possible errors are corrected. In the present proposal we implement the reinforcement functions taking into account all possible values of the input output ports and their complements giving a total of 2^{2N} states, being N the number of ports. The reinforcement functions are implemented to correct for possible faults in the states. The examples of a NOT gate and a NAND gate are given in this section.

A. NOT gate

In a NOT gate, with 2 ports, 16 possible states are considered corresponding to all the combinations of input (x_i, x_{ic}) and output (x_o, x_{oc}) variables. This is shown in Table II. The implementation here proposed is based on reinforcement functions that correct an invalid state by forcing the nearest correct state using the Hamming distance. For example, the state 0010 (S2) should be corrected to the nearest valid state, which is 0110 (S6). With this information, feedback functions are obtained. Incorrect states with more than one faulty variable (states {0000, 0011, 0110, 1010, 1100, and 1111} in Table II) are equidistant to both valid states, and therefore the resulting state cannot be determined. This is expressed as "x" in the truth table to implement the reinforcement functions.

The synthesis of the feedback function was made using Petrick's method [6] with the information in Table II. Its method obtains all the possible minimized functions, which it allows to apply additional optimization strategies, such as to choose the one with the least number of gates. Equations 1 to 4 are the reinforcement functions for the variables of the NOT gate.

TABLE II
POSSIBLE STATES AND CORRECTION BY THE REINFORCEMENT FUNCTION.

Current state						Corrected state				
x_i	x_{ic}	x_o	x_{oc}	state	Dec.	x_i	x_{ic}	x_o	x_{oc}	Dec.
0	0	0	0	I	0	x	x	x	x	x
0	0	0	1	I	1	1	0	0	1	9
0	0	1	0	I	2	0	1	1	0	6
0	0	1	1	I	3	x	x	x	x	x
0	1	0	0	I	4	0	1	1	0	6
0	1	0	1	I	5	x	x	x	x	x
0	1	1	0	V	6	0	1	1	0	6
0	1	1	1	I	7	0	1	1	0	6
1	0	0	0	I	8	1	0	0	1	9
1	0	0	1	V	9	1	0	0	1	9
1	0	1	0	I	10	x	x	x	x	x
1	0	1	1	I	11	1	0	0	1	9
1	1	0	0	I	12	x	x	x	x	x
1	1	0	1	I	13	1	0	0	1	9
1	1	1	0	I	14	0	1	1	0	6
1	1	1	1	I	15	x	x	x	x	x

I: invalid state; V: valid state; x don't care for Petrick's method.

$$x_i = \overline{\overline{\overline{(x_{ic} \cdot \overline{x_{oc}})} \cdot \overline{(x_i \cdot \overline{x_{ic}})}}} \quad \textbf{(1)}$$

$$x_{ic} = \overline{\overline{(\overline{x_i} \cdot x_o)} \cdot \overline{(x_{ic} \cdot \overline{x_{oc}})}} \quad \textbf{(2)}$$

$$x_o = \overline{\overline{(\overline{x_i} \cdot x_o)} \cdot \overline{(x_{ic} \cdot \overline{x_{oc}})}} \quad \textbf{(3)}$$

$$x_{oc} = \overline{\overline{(x_{ic} \cdot \overline{x_{oc}})} \cdot \overline{(x_i \cdot \overline{x_{ic}})}} \quad \textbf{(4)}$$

It is worth noting that the reinforcement functions only aims to correct one error. Figure 5 shows the reinforcement of valid states S6 and S9: any state with one error goes to the corresponding valid states. States with two errors get directed to a valid state arbitrarily (S3, S10), or they eventually reach state S0, where they get stuck until a new input forces a different state. However, in the presence of large random noise, it is expected that such an erroneous state lasts a very short time.

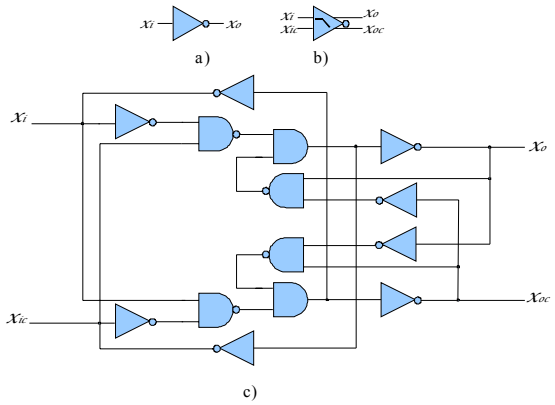


Figure 4. NOT gate logic, a) Classical symbol, b) Symbol for the new proposal, c) Schematic view for our proposal.

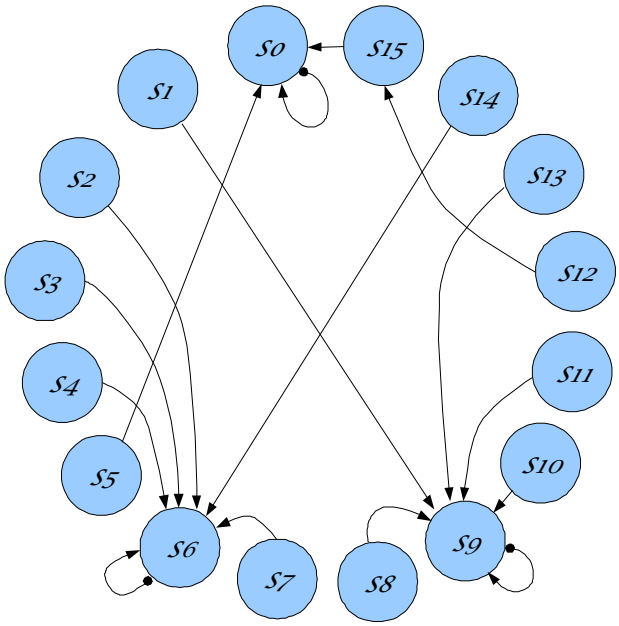


Figure 5. State transitions forces by the reinforcement functions.

B. *NAND gate*

In the case of a 2-input NAND gate, the combinations of input (x_a, x_{ac}, x_b, x_{bc}) and output (x_o, x_{oc}) ports give a total of 64 possible states, of which 4 are valid combinations and the rest have at least one fault. Again, the feedback functions are chosen as to correct a faulty state by reinforcement to the nearest valid state. By means of Petrick method equations 5-10, are obtained where 5-8 are the feedback values that are consistent with the input values, 9 and 10 are the outputs of the system. Figure 6 shows the resulting schematic for the NAND gate.

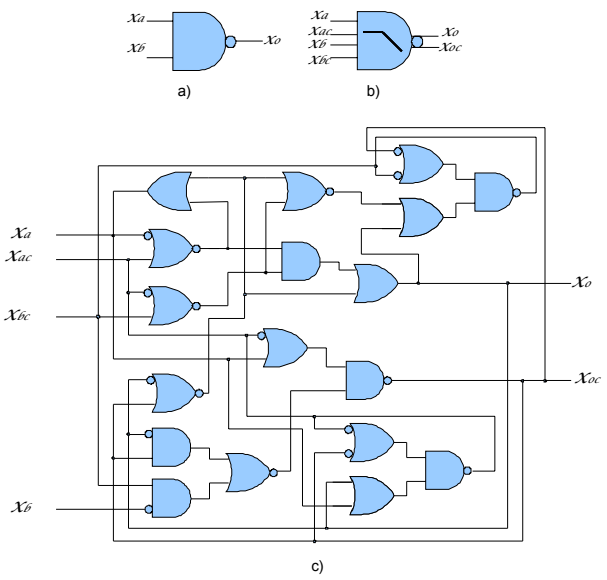


Figure 6. Standard and Our NAND logic function, a) NAND standard symbol, b) our NAND symbol, and c) NAND gate implementation.

$$x_a = (\overline{x_o} \cdot x_{oc}) + (x_a \cdot \overline{x_{ac}}) \quad \textbf{(5)}$$

$$x_{ac} = (\overline{x_a} \cdot \overline{x_{oc}}) + (x_a \cdot x_o) \quad \textbf{(6)}$$

$$x_b = (\overline{x_o} \cdot x_{oc}) + (x_b \cdot \overline{x_{bc}}) \quad \textbf{(7)}$$

$$x_{bc} = (\overline{x_b} \cdot \overline{x_{oc}}) + (x_b \cdot x_o) \quad \textbf{(8)}$$

$$x_o = (x_o \cdot \overline{x_{oc}}) + (\overline{x_b} \cdot x_{bc}) + (\overline{x_a} \cdot x_{ac}) \quad \textbf{(9)}$$

$$x_{oc} = (\overline{x_o} \cdot x_{oc}) + (x_a \cdot \overline{x_{ac}} \cdot x_b \cdot x_{bc}) \quad \textbf{(10)}$$

IV. Simulation Results

In order to show the improved robustness of our approach, we implemented a NOT gate and a NAND gate with three methods: standard CMOS, MRF Reinforcer, and our approach, all three in a 90nm CMOS technology. A V_{DD} in the range of 0.65 V to 0.45 V are the current ITRS predictions [7], and therefore in this work we use a V_{DD} of 0.5 V.

As we are interested in evaluating the robustness, the inputs are chosen as digital (0 to V_{DD}) with an additive Gaussian random noise of mean 0V and standard deviation of 0.6V rms. Uncorrelated noise sources are applied to each of the inputs, while the outputs are left unforced.

The obtained transient waveforms are shown in Figures 7 and 8 (NOT and NAND gates respectively). It can be seen how our proposal has a very good Signal to Noise Ratio (SNR) especially in the NAND gate case, where the standard CMOS has a nearly useless output signal, and the MRF proposal presents spurious oscillations.

SNR values at input and output are given in Table III, and the results show an excellent behavior of our proposal.

TABLE III

Comparative static, MRF and our techniques

Circuit	*Logic*	*In*	*Out*	*# Transistors*	*Over-Head (N times)*	*SNR input (dB)*	*SNR output (dB)*
Inverter	static	1	1	2	1.0	-1.58	1.93
	MRF	4	4	20	10.0		15.91
	Our	4	4	48	24.0		39.97
NAND	static	2	1	4	1.0		*N.A*
	MRF	4	2	60	15.0		*N.A*
	Our	4	2	78	19.5		26.50

In: number of inputs, Out: number of outputs.

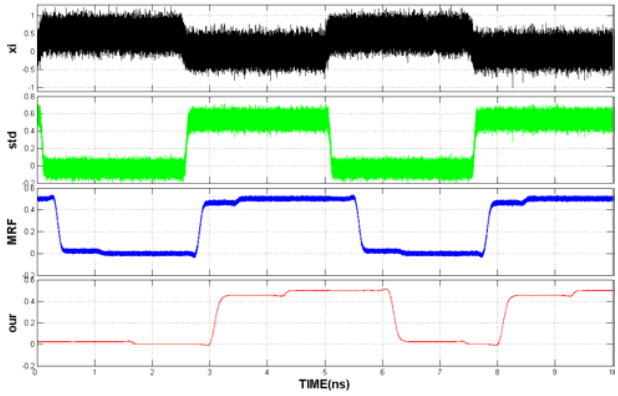


Figure 7. Simulation results for inverter standard CMOS, MRF and our proposal.

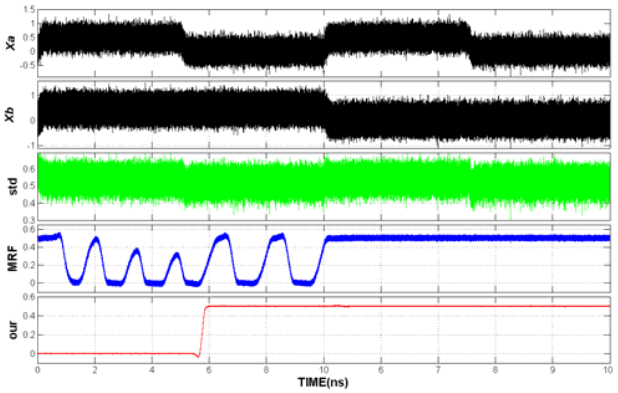


Figure 8. Simulation results for NANDs standard CMOS, MRF and our proposal.

V. Conclusions and Future Work

The results shown in the previous section demonstrate an excellent tolerance to noise of the present approach compared to the previous proposals. The improvement with respect to MRF Reinforcement is due to a better evaluation of the reinforcement functions in terms of all possible errors.

This improvement is obtained at the expense of a large overhead in area and timing that must be carefully evaluated. This area and timing penalties would be prohibitive in today's noise environment and technologies. The field of application of this new technique is therefore beyond the conventional CMOS circuits. We see two possible scenarios where this proposal could be of interest. One scenario is aimed at extremely high Noise-to-Signal ratios where conventional CMOS logic cannot be used and therefore the penalties are affordable, for instance aerospace technology. The other possible scenario is one of beyond-CMOS devices with a large degree of behaviour uncertainty, and where the small scale of the devices makes the area overhead again affordable.

In any case, as future work the overheads will be more thoroughly evaluated and minimized. Another interesting aspect is the investigation of tolerance to hard and other type of soft defects apart from noise. Also, the combination of several gates must be studied to assess possible design difficulties due to the interaction of outputs and reinforcement functions of the next gate.

Acknowledgments

This research work has been supported by the Spanish Ministry of Science and Innovation (MICINN) through the project TEC2008-01856 with the additional participation of FEDER founds and CONACyT Mexico under grant 164013.

The group of research is considered a consolidated group by the MICINN.

References

[1] J. Von Neumann, "Probabilistic logics the Synthesis of Reliable Organisms from Unreliable Components", Automata Studies, C. E. Shannon and J. McCarthy, eds., Princeton Univ. Press, 1956, pp. 43-98.

[2] W. H. Pierce, "Failure-Tolerant computer design", Academic Press, 1965.

[3] J. G. Tryon, "Quadded Logic" Redundancy techniques for computing systems, R. H. Wilcox and W. C. Mann, eds., Spartan books, pp. 205-228, 1962.

[4] R. Kindermann, J. L. Snell, Markov Random Fields and their Applications, American Mathematical Society, 1980.

[5] K. Nepal, R. I. Bahar, J. Mundy, W. R. Patterson, and A. Zaslavsky, "Designing logic circuits for probabilistic computation in the presence of noise," DAC 2005, June 13-17, 2005, Anaheim, California, USA.

[6] Giovanni De Micheli., "Synthesis and optimization of digital circuits", McGraw Hill 1994, Chapter 8, ISBN: 0-07-016333-2.

[7] International Technology Roadmap for Semiconductors, online. Available http://public.itrs.net .

Turtle Logic: A new probabilistic design methodology of nanoscale digital circuits

Lancelot Garcia-Leyva
Universidad Autónoma de Tlaxcala
Facultad de Ciencias Básicas, Ingeniería y Tecnología.
C/Apizaquito s/n, Apizaco, 90300
Tlaxcala, México.
Email: lancelot@eel.upc.edu

Antonio Calomarde, Francesc Moll and Antonio Rubio
Universitat Politecnica de Catalunya
Department of Electronic Engineering
C/Jordi Girona, 31, 08034, Barcelona, Spain.
Email: {calomarde, moll}@eel.upc.edu,
antonio.rubio@upc.edu

***Abstract*— As devices and operating voltages are scaled down, future circuits will be plagued by higher soft error rates, reduced noise margins and defective devices. A key challenge for the future technologies is to retain circuit reliability in the presence of faults and noise. The Turtle Logic (TL) is a new probabilistic logic method based on port redundancy and complementary data, oriented to emerging and beyond CMOS technologies. The TL is a technology independent method, which aims to improve tolerance to errors due to noise in single gates, logic blocks or functional units. The TL operation is based on the consistency relation of redundant inputs. In case of discrepancy, the output of the system keeps the previous value, therefore avoiding the propagation of incorrect inputs. Simulations show an excellent performance of TL in the presence of large random noise at the inputs, as well as intrinsic noise (thermal noise and flicker noise) and shot noise in the power source.**

I. Introduction

With the progress on VLSI process technology, the design complexity and the transistor density in system increases rapidly, causing that the power consumption and power density in system rise with the same trend. The size of CMOS devices is scaled down to the nanoscale level where interferences (soft errors), becoming significant, affect the VLSI circuit performance [1]. Future electronic devices are expected to operate at lower voltage supply to save power, especially in ultimate and new technologies. The resulting reduction of logic levels approaches the thermal noise limit, and consequently signal to noise margins are reduced, exposing computations to higher soft-error rates [1].

In order to design reliable circuits with unreliable components, new design techniques must be introduced. The problem of designing reliable systems with unreliable components traces back to Von Neumann, who proposed the N-tuple Modular Redundancy (NMR) technique [2]. In the NMR each gate, logic block or functional unit is replicated N times and the final output is obtained through a majority voting circuit, where the majority voting gate need to be perfect and free of faults. Several approaches have been proposed based on the NMR such as the Triple Modular Redundancy (TMR) or the Cascaded TMR (CTMR) [3].

Additional proposals have appeared in the literature addressing the problem from the point of view of noise tolerance instead of exclusively considering hard defects. For instance, the so called Probabilistic Logic based on Markov Random Field theory (MRF) [4], identifies valid and invalid states and derives a logic implementation which tries to reinforce valid states in order to increase their probability and decrease the probability of erroneous states. Another approach is the Probabilistic CMOS (PCMOS) [5], where its principal objective is to develop and comprehensively characterize probabilistic CMOS circuits that may be used to build energy efficient computing platforms, where reliability of the circuit is not necessarily the objective.

Our proposal is based on the scenario that all components are noisy and hence they may fail. In order to increase circuit reliability, we propose to increase the number of input and output ports in each module, which is a form of data redundancy. This new proposal is different from the Von Neumann NMR approach in two aspects: that each module is not replicated n times, but instead, the module is redesigned with I/O ports replicated n times to increase the reliability, and voting circuit is not needed.

The structure of the paper is as follows: in Section II, the basic concepts and principles of error probability are introduced, as well as the error caused by noise in one digital node. In section III, Port Redundancy as a method to improve the reliability of a module under noise scenario is presented. In Section IV our proposal called Turtle Logic is introduced, explaining the implementation of a NOT, AND-NAND, OR-NOR and XOR-XNOR gates as examples. Section V presents simulation results of the different approaches using a 90nm CMOS process technology. Finally, the conclusions are presented in Section VI.

II. Probabilistic formulation

In ultra-low V_{DD} CMOS scenario and future technology, the noise margin will be too small, and therefore soft errors due to all kinds of noise sources become crucial. For simplicity,

This research work has been supported by the Spanish Ministry of Science and Innovation (MICINN) through projects TEC2008-01856 and PLE2009-0024 (ENIAC MODERN project), with the participation of FEDER funds and "Plan E" funds, and Government of Mexico under grant 164013 CONACyT and UATLX-235 PROMEP. The group of research is considered a consolidated group by the MICINN.

978-1-4244-7772-2/10/$26.00 ©2010 IEEE

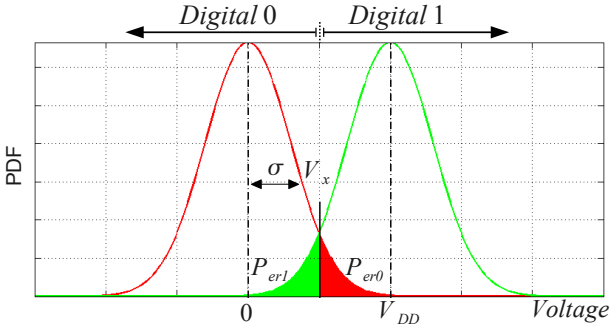


Fig. 1. The Probability Density Function for digital values with $\mu = 0V$ and $\mu = V_{DD}$ voltages in a digital node, under influence of White Gaussian Noise.

and without loss of generality, we model signals as two-logic levels, with additive white Gaussian uncorrelated noise, with mean μ, standard deviation σ and variance σ^2.

A. Error caused by noise in one digital node

Fig. 1 shows the voltage probability distribution in a noisy digital node. It can be described as two Gaussian distributions centered around low and high voltages (0 and VDD respectively). An error can be produced when a logic 1 is interpreted as a logic 0 and vice versa. The respective probabilities of such cases are denoted as P_{er1} and P_{er0}, they are obtained integrating the logic 1 and logic 0 PDF functions beyond the logic threshold (denoted as Vx in Fig. 1).

Assuming that logic 1 and logic 0 are equally probable, the probability that the data in a node is incorrect is given by:

$$P_e = \frac{P_{er0} + P_{er1}}{2} \tag{1}$$

III. Port Redundancy

In this section, we analyze the reliability improvement of input and output ports redundancy (PR) to the modules. In Fig. 2, we show how a module can have $(n-1)$ replicated ports, where the conventional differential logic is a particular case $(n = 2)$. The respective replicated ports can be either equal or complementary to the original port. In general, port redundancy has three different scenarios, which probability can be calculated as follow:

1) Correct scenario (CS): All the inputs logic values are interpreted correctly. The probability of this correct scenario when n ports are considered is given by equation (2).

$$P_{cn} = (1 - P_e)^n \tag{2}$$

2) Discrepant scenario (DS): One or several ports have voltage values which cause an incorrect interpretation (logic 0 instead of intended logic 1 or vice versa). The probability that this happens is given by equation (3).

$$P_{dn} = \sum_{k=1}^{n-1} \frac{n!}{k!(n-k)!} (1 - P_e)^{n-k} P_e^k \tag{3}$$

3) Error scenario (ES): All the ports are misinterpreted, as a logic 1 instead of an intended logic 0 or vice versa. In this case, the port interpretation is apparently coherent (they keep their internal coherence). The probability of this scenario is given by equation (4).

$$P_{en} = P_e^n \tag{4}$$

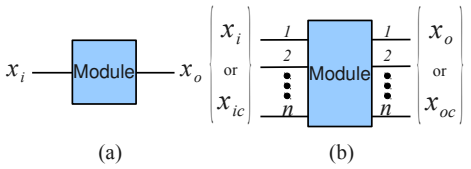


Fig. 2. Port redundancy. (a) Single gate with one input and one output port, (b) Single gate with $(n-1)$ replicated ports.

The specific case of a NOT gate with PR1 (PR with $n = 1$) will be considered in the next section.

A. Probability scenarios for a NOT gate with PR1

The probabilities for respective scenarios are given by:

1) CS: The values of the both ports are correct, Fig. 3(a).

$$P_{c2} = (1 - P_e)(1 - P_e) = (1 - P_e)^2 \tag{5}$$

2) DS: Only one value of the two ports is incorrect, Fig. 3(b).

$$P_{d2} = 2P_e(1 - P_e) \tag{6}$$

3) ES: Both ports are incorrect, Fig. 3(c).

$$P_{e2} = P_e P_e = P_e^2 \tag{7}$$

Each redundant port is generated by a different driver, so that the noise can be considered statistically independent.

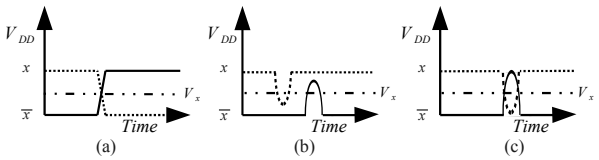


Fig. 3. Scenarios for CMOS single gate with one complementary replicated port, (a) Both signal are correct (P_{c2}), (b) One of both signals is incorrect (P_{d2}) and, (c) Both signal are incorrect (P_{e2}).

B. Analysis of error reduction with port redundancy (PR)

The error probability (P_{en}) follows an exponential relationship with the number of replicated ports as shown in equation (4). Fig. 4 shows the results of error probability for a single gate with redundancies from 0 to 9, for an SNR from -3 to 16 dB. As expected, if the number of replicated ports is increased, the error probability is reduced. For the specific case of PR1 with SNR=-3dB ($\sigma = 0.5$ Vrms and $V_{DD} = 0.5$ V), the decrease in error probability is 34.57%. When the PR is increased further to 2, 3 and 4, the corresponding decrease in error probability is smaller each time, with decrements respect

to the previous PR of 11.95%, 4.13%, 1.41%, 4.91e-3%, respectively, so that for PR beyond 4 a negligible decrement in error probability is observed. For SNR greater than -3dB, the advantage of redundancy 1 over redundancy 2, 3, etc. is lower every time. Conversely, the cost in hardware of a digital circuits with PR is highly increasing. Therefore, a redundancy of one (PR1 – increasing in one the number of ports) is considered in this work, as a good balance between reliability and overhead.

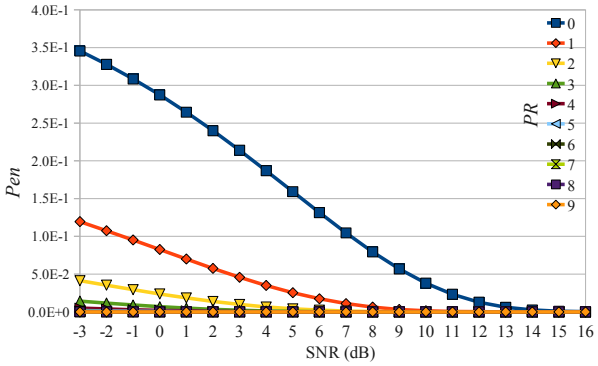


Fig. 4. Error probability for a single gate with 0-9 port redundancies.

IV. New Design for Robust Digital Circuits

The design proposal presented in this paper consists in that logic functions are implemented such that the consistency of all redundant inputs is checked. In case of discrepancy, the output of the gate holds its previous values, therefore avoiding the propagation of incorrect inputs to the outputs. We call this concept Turtle Logic (TL). To show the operating of TL design, we use a NOT gate with PR1 and complementary data. The circuit obtained is shown in Fig. 5, and this TL NOT gate has two possible cases:

- When the input ports x_i and x_{ic} have complementary values, for instance, $x_i = 0$ and $x_{ic} = 1$ or vice versa, the TL NOT identifies the inputs as correct values and forces the complementary output ports driving the appropriate values according to the inverter function ($x_o = 1$, $x_{oc} = 0$).
- When the inputs have the same value ($x_i = 0$, $x_{ic} = 0$ or $x_i = 1$, $x_{ic} = 1$), TL NOT gate detects these values as an error. Then, the output ports of the system are forced to hold their previous values.

It has to be noted that when both inputs take simultaneously an incorrect value due to noise, the TL gate admits them as correct values, thus yielding an incorrect output. However, the probability of this simultaneous event is very low when there is a statistical independence of the noise in redundant ports, as shown in equation (4) and Fig. 4.

Figs. 6 and 7 show a NAND and an XOR gate using the TL methodology described before, and the data about the overhead in gates of their respective implementations is shown in Table I.

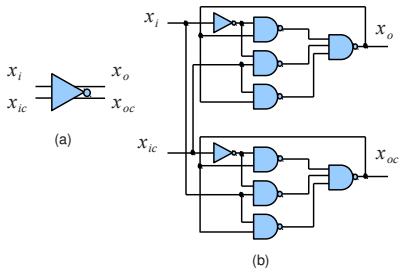


Fig. 5. NOT gate using TL with PR1. (a) Symbol, and (b) Schematic view.

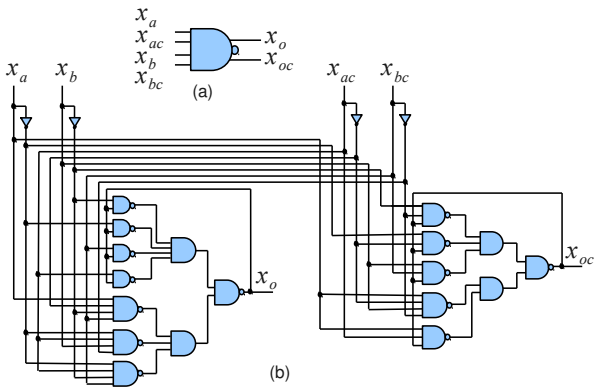


Fig. 6. NAND gate using TL with PR1. (a) Symbol, and (b) Schematic view.

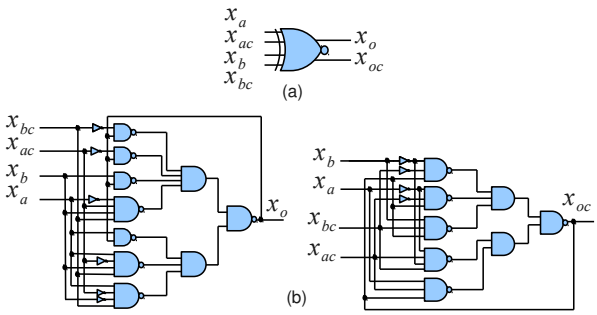


Fig. 7. XOR gate using TL with PR1. (a) Symbol, and (b) Schematic view.

V. Simulation Results

In order to validate the improved robustness of the TL proposal Fig.5(b), we implemented a NOT gate with three methods: standard CMOS, MRF [4] and TL. The results shown in Fig. 8 are the result of SPICE simulations, using 90nm technology device models with a V_{DD} of 0.5V at temperature of 100 ^{o}C. Internal noise was generated using transient-noise option of the Spectre circuit simulator, which generates thermal noise and flicker noise to the channel of each transistor, and shot noise to each power supply at each time step. The noise generated to each transistor and power source were amplified 100 times respect to intrinsic noise of the 90nm technology, in order to emulate the behaviour of future technology with very noisy components. It is worth noting that nominal V_{DD} for this technology is 1V, but in this work it was decreased to 0.5V to simulate an equivalent

TABLE I

COMPARATIVE CMOS, MRF AND TURTLE LOGIC TECHNIQUES FOR A NOT GATE, USING ALTERNATING 5,000 1'S AND 0'S, WITH PERIOD T OF 4NS.

Logic		# Inputs	# Outputs	# Gates	ER x_o	ER x_{oc}
NOT	Static	1	1	1	8,298	N. A.
	MRF	2	2	8	1,580	2,850
	TL	2	2	10	71	73
AND-NAND	Static	2	1	2/1	5,778	10,864
	TL	4	2	18	596	939
OR-NOR	Static	2	1	2/1	4,061	13,351
	TL	4	2	18	930	1,163
XOR-XNOR	Static	2	1	5/5	12,313	14,545
	TL	4	2	18	1,683	1,867

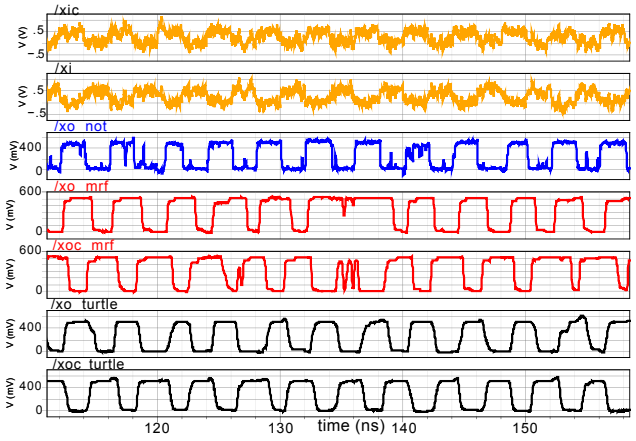


Fig. 8. Simulation results for a NOT gate using Standard CMOS, MRF and Turtle logic.

environment in future technologies, where thermal and flicker noise will have greater relevance. The input noisy signals were generated with Additive White Gaussian Noise (AWGN) modules implemented using Verilog-A, with 0V mean and standard deviation of 0.5 Vrms to both logic levels 1 and 0, using different seed for each input. Uncorrelated noise sources are applied to each of the inputs, while the outputs of the device under test have identical NOT gates as load.

The simulated transient waveforms are shown in Fig. 8, where *xi* and *xic* are common input signals to all devices. *xo_not*, *xo_mrf*, *xoc_mrf*, *xo_turtle* and *xoc_turtle* are the output signals for standard CMOS, MRF and Turtle Logic, respectively. It can be seen that Turtle Logic has a very good robustness in an environment where noise levels reach similar magnitude as V_{DD}.

Table I shows the overhead as well as the noise error robustness of standard CMOS, MRF gates and Turtle logic approaches for the NOT, AND-NAND, OR-NOR and XOR-XNOR gates. As the reliability parameter to measure and compare the noise error robustness of these approaches we use the error ratio (ER) factor, shown in the respective column of table I. The meaning of this ER factor depends on the number of spurious signals observed at the outputs of the circuits after a periodic sequence of 0's and 1's is applied at their inputs. The period of the sequence is 4 ns for the NOT gates, and for the other gates, one of the inputs has 4 ns period and the other has 8 ns period in order to have all the possible combinations. A spurious signal (SS) is any signal which has an unexpected voltage in a time interval [6]. An SS is counted in the results table as one ER when it has an amplitude greater than $V_x \pm 0.1V_{DD}$ (for a 0 logic and 1 logic, respectively) and width at least 10% of the signal period ($T = 0.4$ ns in our example).

As can be seen in Table I, in the case of a NOT gate we found 8,298 spurious for standard CMOS, 1,580 for the MRF, and 71 for TL. TL NOT gate has a much lower ER than Standard CMOS or MRF, therefore demonstrating that TL is more robust in extremely noisy environments.

VI. CONCLUSIONS

The results shown in the previous section demonstrate an excellent tolerance to high noise of the present approach compared to the previous proposals. This paper shows how increasing port redundancy decreases the error probability. This result in addition with Turtle Logic is the basis for a new way of designing logic blocks which are tolerant to a high level of noise, under the assumption of statistical independence of noise (the addition of thermal noise, flicker noise and coupled noise) in future and new technologies.

This improvement is obtained at the expense of a large overhead in area and timing that must be carefully evaluated. It can be argued that the obtained area and timing penalties would be prohibitive in today's noise environment and technologies. However, this level of overhead can be acceptable in future technologies beyond CMOS, where the scenario of low SNR (extremely high noise) considered in this paper will need reliable circuits. In this conditions, the design strategy here proposed will be an adequate option.

REFERENCES

[1] [Online]. Available: http://www.itrs.net/Links/2009ITRS/Home2009.htm

[2] J. V. Neumann, *Probabilistic Logics the Synthesis of Reliable Organisms from Unreliable Components*, C. E. S. Automata Studies and e. J. McCarthy, Eds. Automata Studies, C. E. Shannon and J. McCarthy, eds., 1956.

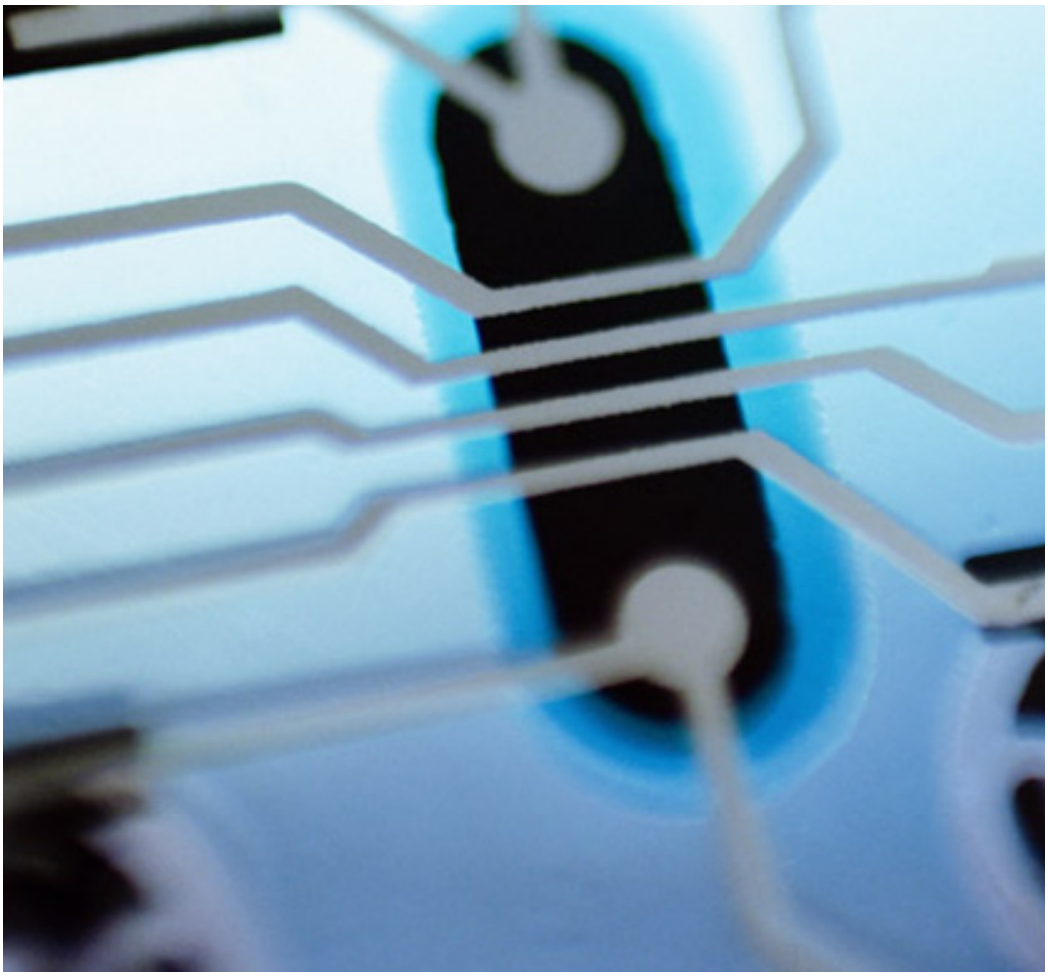
[3] W. H. Pierce, *Failure-Tolerant computer design*. Academic Press., 1965.

[4] K. Nepal, R. Bahar, J. Mundy, W. Patterson, and A. Zaslavsky, "Designing logic circuits for probabilistic computation in the presence of noise," in *Design Automation Conference, 2005. Proceedings. 42nd.*, June 2005, pp. 485–490.

[5] P. Korkmaz, "Probabilistic CMOS (PCMOS) in the nanoelectronics regime," Ph.D. dissertation, Georgia Institute of Technology, Dec. 2007.

[6] F. Moll and A. Rubio, "Spurious signals in digital CMOS VLSI circuits: a propagation analysis," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, vol. 39, no. 10, pp. 749–752, Oct 1992.

Proceedings of the 1st Barcelona Forum on Ph.D. Research in Electronic Engineering



October 16th, 2009

Universitat Politècnica de Catalunya
Campus Nord—Edifici Vèrtex

New Methodology of Digital IC Design in extremely high noise and low voltage Scenarios

Author: Lancelot García Leyva
Thesis Advisors: Antonio Rubio, Francesc Moll and Antonio Calomarde

I. Introduction

With the progress on VLSI process technology, the design complexity and the transistor density in SoC increase rapidly, leading to the power consumption and power density in SoC designs rising with the same trend. The size of CMOS devices is scaled down to the nanoscale level, noise interferences becoming significantly affect the VLSI circuit performance. Future electronic devices are expected to operate at lower voltage supply to save power, especially in ultimate and new technologies.

The noise magnitude every day is more important respect the signal magnitude. Noise is a purely random signal, the instantaneous value and/or phase of the waveform cannot be predicted at any time, therefore a probabilistic-based approach is more suitable to handle signal errors than the conventional deterministic circuit design. Noise can either be generated internally in the device, from its associated passive components, or superimposed on the circuit by external sources. The main noise type that affect a digital system is the thermal and flicker noise. The resulting reduction of logic levels approaches to the thermal noise limit, and consequently reduce signal to noise margins will exposing computation to higher soft-error rates. Therefore, circuit designers can no longer assume that future circuits will have error-free operation.

Recently, as the reliability problems associated to technology scaling have become apparent, new proposals have appeared in the literature addressing the problem from the point of view of noise tolerance instead of exclusively considering hard defects. Among the new proposals, there is the so called Probabilistic Logic based on Markov Random Field theory (MRF CMOS [1]) which identifies valid and invalid states, and derives a logic implementation trying to reinforce valid states in order to increase their probability and decrease the probability of erroneous states.

Our proposal is based in detect errors by redundancy of data and correct the errors detected.

II. Proposal

The aim of this work is to increase the robustness of the devices, by introducing redundancy in them by mean of duplicated the number of signals x , adding the complementary data of $\bar{x}$. This allows the system has more information to detect and correct errors in its inputs and outputs, when the system is under a big amount of noise (thermal and flicker noise mainly).

Our idea is take into account the complete set of possible states that reinforce the correct states. In the present proposal we implement the reinforcement functions taking into account all possible values of the input and output ports and their complements giving a total of 2^{2N} states, being N the number of ports. The reinforcement functions are implemented to correct possible faults in the states. The examples of a NOT gate and a NAND gate are given in next sections.

II.A. NOT gate

In a NOT gate, with 2 ports, there are 16 possible states to all the combinations of input (x_i, x_{ic}) and output (x_o, x_{oc}) variables, this is shown in Table 1. The implementation here proposed is based on reinforcement functions that correct an invalid state by forcing the nearest correct state using the Hamming distance. For example, the state 0010 should be corrected to the nearest valid state, which is 0110. With this information, feedback functions are obtained. Incorrect states with more than one faulty variable (states {0000, 0011, 0110, 1010, 1100, and 1111} in Table 1) are equidistant to both valid states, and therefore the resulting state cannot be determined. This is expressed as "x" in the truth table to implement the reinforcement functions.

Current state						Corrected state				
x_i	x_{ic}	x_o	x_{oc}	state	Dec.	x_i	x_{ic}	x_o	x_{oc}	Dec.
0	0	0	0	I	0	x	x	x	x	x
0	0	0	1	I	1	1	0	0	1	9
0	0	1	0	I	2	0	1	1	0	6
0	0	1	1	I	3	x	x	x	x	x
0	1	0	0	I	4	0	1	1	0	6
0	1	0	1	I	5	x	x	x	x	x
0	1	1	0	V	6	0	1	1	0	6
0	1	1	1	I	7	0	1	1	0	6
1	0	0	0	I	8	1	0	0	1	9
1	0	0	1	V	9	1	0	0	1	9
1	0	1	0	I	10	x	x	x	x	x
1	0	1	1	I	11	1	0	0	1	9
1	1	0	0	I	12	x	x	x	x	x
1	1	0	1	I	13	1	0	0	1	9
1	1	1	0	I	14	0	1	1	0	6
1	1	1	1	I	15	x	x	x	x	x

Table 1. Possible states and correction by the reinforcement function. I - Invalid state, V – valid state, x do not care for Petrick's method.

The synthesis of the feedback function are made using Petrick's method according to the information presented in Table 1. Its method obtains all the possible minimized functions, this allows to apply additional optimization strategies, choosing the one with the least number of gates. Equations 1 to 4 are the reinforcement functions for the variables of the NOT gate and its implementation is shown in the Fig. 1.

$$x_i = \overline{(\overline{x_{ic} \cdot x_{oc}}) \cdot (\overline{x_i \cdot x_{ic}})} \quad \textbf{(1)}$$

$$x_{ic} = \overline{(\overline{x_i \cdot x_o}) \cdot (\overline{x_{ic} \cdot x_{oc}})} \quad \textbf{(2)}$$

$$x_o = \overline{(\overline{x_i} \cdot x_o)} \cdot \overline{(x_{ic} \cdot \overline{x_{oc}})} \quad (3)$$

$$x_{oc} = \overline{(\overline{x_{ic}} \cdot x_{oc})} \cdot \overline{(x_i \cdot \overline{x_{ic}})} \quad (4)$$

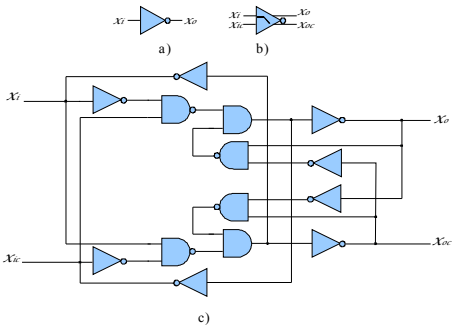


Figure 1. NOT gate logic, a) Classical symbol, b) Symbol for the new proposal, and c) Schematic view for our proposal.

II.B. NAND gate

In the case of a 2-input NAND gate, the combinations of input (x_a, x_{ac}, x_b, x_{bc}) and output (x_o, x_{oc}) ports give a total of 64 possible states, of which 4 are valid combinations and the rest have at least one fault. Again, the feedback functions are chosen to correct a faulty state by reinforcing to the nearest valid state. Fig. 2 shows the resulting schematic for the NAND gate.

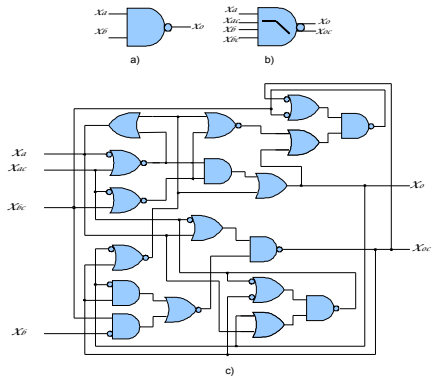


Figure 2. NAND gate logic, a) Classical symbol, b) Symbol for the new proposal, and c) Schematic view for our proposal.

III. Simulation results

In order to show the robustness of our proposal two logic gates was implemented a NOT gate and a NAND gate with three different methods: standard CMOS, MRF reinforcer, and our approach, all three in a *90nm* CMOS technology. A V_{DD} in the range of *0.65* V to *0.45* V are the current ITRS predictions for 2022 year [2], and therefore for show the results in this work we use a V_{DD} of *0.5* V.

The gates logic are evaluating in the inputs with an additive Gaussian random noise of mean *0*V and standard deviation of *0.6*V rms (from *0* to V_{DD}). Uncorrelated noise sources are applied to each of the inputs, while the outputs are left unforced.

The obtained transient waveforms are shown in Figs. 3 and 4 for the NOT and NAND gates, respectively. It can be seen how our proposal has a better Signal to Noise Ratio (SNR) especially in the NAND gate case, where the standard CMOS has a nearly useless output signal, and the MRF proposal presents spurious oscillations.

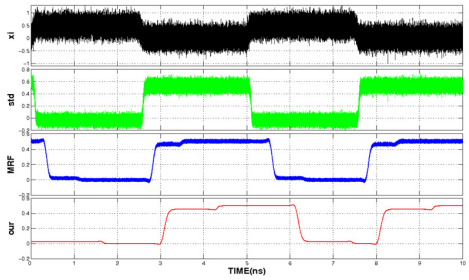


Figure 3. Simulation results for inverter standard CMOS, MRF and our proposal.

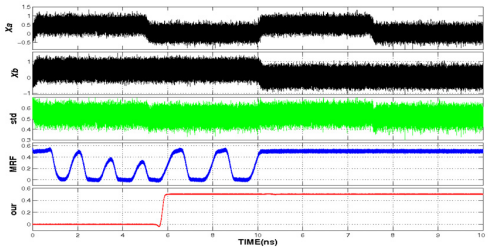


Figure 4. Simulation results for NAND standard CMOS, MRF and our proposal.

IV. Conclusions

The results shown in the previous section demonstrate an better noise tolerance of the present approach compared to the previous proposals. The improvement with respect to MRF Reinforcement is due to a better evaluation of the reinforcement functions in terms of all possible errors.

This improvement is obtained at the expense of a large overhead in area and timing that must be carefully evaluated. This area and timing penalties would be prohibitive in today's noise environment and technologies. The target field of application of this new technique is therefore the ultimate and new technologies. One scenario is beyond-CMOS devices with a large degree of behavior uncertainty, and where the small scale of the devices makes the area overhead again affordable.

V. Acknowledgments

This research work has been supported by the Spanish Ministry of Science and Innovation (MICINN) through the project TEC2008-01856 with the additional participation of FEDER founds and CONACyT Mexico under grant 164013. The group of research is considered a consolidated group by the MICINN.

VI. References

[1] K. Nepal, R. I. Bahar, J. Mundy, W. R. Patterson, and A. Zaslavsky, "Designing logic circuits for probabilistic computation in the presence of noise," DAC 2005, June 13-17, 2005, Anaheim, California, USA.

[2] International Technology Roadmap for Semiconductors, online. Available http://public.itrs.net .

Proceedings of the Barcelona Forum on Ph.D. Research in Communications, Electronics and Signal Processing

Edited by
José Luis González and Alejandro Rodríguez
Universitat Politècnica de Catalunya
Barcelona, October 2010
ISBN: *978-84-7653-495-3*

For more information on the Doctoral Programs on Electronic Engineering and Communications and Signal Theory, please, visit our websites at
http://doctorat.eel.upc.edu
http://www.tsc.upc.edu/doctorat

Turtle logic: Novel IC digital probabilistic design methodology

Author: Lancelot Garcia-Leyva, lancelot@eel.up.edu
Thesis Advisors: Antonio Rubio, Francesc Moll and Antonio Calomarde

I. Introduction

With the progress on VLSI process technology, the design complexity and the transistor density in system increases rapidly, causing that the power consumption and power density in system rise with the same trend. The size of CMOS devices is scaled down to the nanoscale level where interferences (soft-errors), becoming significant, affect the VLSI circuit performance [1]. Future electronic devices are expected to operate at lower voltage supply to save power, especially in ultimate and new technologies. The resulting reduction of logic levels approaches the thermal noise limit, and consequently signal to noise margins are reduced, exposing computations to higher soft-error rates [1]. In other words, the future circuits will be in a scenario where all devices may fail due to soft-error produced by trend of low SNR.

In order to design reliable circuits with unreliable components, novel design techniques have been introduced. The problem of designing reliable systems with unreliable components traces back to Von Neumann, who proposed the N-tuple Modular Redundancy (NMR) technique [2]. Additional proposals have appeared in the literature addressing the problem from the point of view of noise tolerance. For instance, the approach based on Markov Random Field theory (MRF) [3]. [4] Take in account the Hamming distance for build basic logic gates focus to high noise and low voltage scenarios.

Therefore, our proposal is based on the assumption that the devices in new and future technologies will be not perfect, noisy and hence they might fail.

II. Probabilistic formulation

In low power supply scenario and future technology, the noise margin will be too small, and therefore soft errors due to all kinds of noise sources become crucial.

In our proposal, the technology independent has been considered, and only we take into account noise level and duration which propagate through logical primitives [5] with probability P_e. For simplicity, and without loss of generality, the signals may be modeled as two-logic level, with additive white Gaussian noise $f(x)$, no correlated, with mean μ, standard deviation σ and variance σ^2.

II.A. Error caused by noise in one digital node

Figure 1 shows the voltage probability distribution in a noisy digital node. It can be described as two Gaussian distributions centered on low and high voltages (0 and V_{DD} respectively). An error can be produced when logic 1 is interpreted as logic 0 and vice versa. The respective probabilities of such cases are denoted as P_{e1} and P_{e0}.

Assuming that logic 1 and logic 0 are equally probable, the probability that the data in a node is incorrect P_e is given by the addition of P_{e1} plus P_{e0} divided by two.

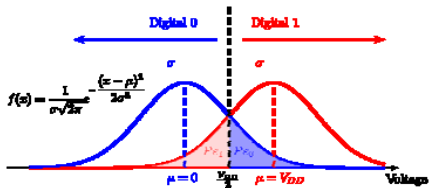


Figure 1. The Probability Density Function for digital values with $\mu = 0$ (V) and $\mu = V_{DD}$ (V) in a digital node, under influence of White Gaussian Noise.

III. Port redundancy

In this section, the improvement of reliability of digital modules using ports redundancy (PR) is presented. The Figure 2 shows how a module with one input and one output may have $(n - 1)$ replicated ports, and where the conventional differential logic is a particular case $(n = 2)$. The respective replicated ports can be either equal or complementary to the original port as can be seen in the Figure 2(b).

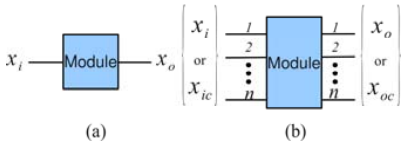


Figure 2. Port redundancy. (a) Module with one input and one output port (for instance, a traditional NOT gate), (b) Module with $(n - 1)$ replicated ports.

In general, a port redundancy has three different scenarios, with probabilities defined as:

1. Correct scenario: the values of all logic inputs are interpreted correctly. The probability is $(1-Pe)^n$.
2. Discrepant scenario: One or several ports have Voltage values which causes an incorrect interpretation (logic 0 instead of intended logic 1 or vice versa). The probability is given by the permutation sum of Pe and $(1-Pe)$.
3. Error scenario: All the ports are misinterpreted and it has a probability of P_e^n.

III.A. Analysis of error reduction with port redundancy

The error probability (P_{emi}) depends exponentially on the number of replicated ports. The Figure 3 shows the results of error probability for a single gate with redundancies from 0 to 9, for a SNR from 1 to 20 dB. As expected, if the number of replicate ports is increased, the error probability is reduced. For the specific case of Redundancy of 1 with SNR=1dB, the decrease in error probability is of 20.44%. When further increasing the in 2, 3 and 4 times, the corresponding decrease in error probability is smaller each time, with decrements respect to before redundancies of 8.26%, 5.9%, 1.68%, 0.49%, respectively, so for redundancies beyond of 4 times gives a negligible decrement in error probability. For SNR greater than 1dB, the advantage of redundancy one over redundancy two, three, etc. is lower every time. Inversely, the cost in hardware of a

digital circuit with port redundancies is higher every time. herefore, a redundancy of one is appropriated tradeoff etween relia ility and overhead.

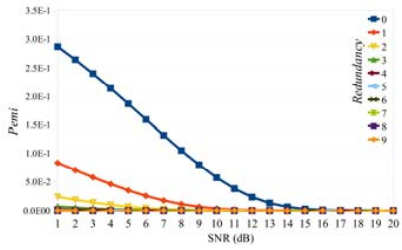


Figure 3. Error pro a ility for a single gate with 0-9 port redundancies.

IV. Turtle Logic: redundant digital design for ICs

he design proposal presented in this work consists in replicate the ports one time using the complementary data for the devices has to reject noise in common mode. he logic functions are implemented such that the consistency of all redundant inputs is checked. In case of discrepancy, the output of the gate holds its previous values, hence avoiding the propagation of incorrect inputs to the outputs. We call this concept urtle Logic (L) ecause the circuit mimic the of the turtles in the nature, for example, when it is in danger environment (noise for the circuits), it is remain in stand y ut does not died, and when the danger disappear the turtle continue with its normal live (for the case of the circuits, they continue with the correct functionality). o show the operating of L design, we use a NO gate with port redundancy of 1 and complementary data.

he circuit o tained at gate level is shown in Figure. 4 which has two possi le cases:

- When the input ports have complementary values (x_i != x_{ic}), then, the L NO identifies the inputs as correct values and forces the complementary output ports driving the appropriate values according to the NO function $x_o = not(x_i)$, and $x_{oc} = not(x_{ic})$.
- When the inputs have the same value ($x_i = x_{ic}$), L NO gate detects these values as an error. here y, the output ports of the system are forced to hold their previous correct values.

It has to e noted that when oth inputs take simultaneously an incorrect value due to noise, the L gate admits them as correct values, thus yielding an incorrect output. Nevertheless, the pro a ility of this simultaneous event is very low when there is a statistical independence of the noise in redundant ports, as must een deduced of the Figure. 3.

V. Simulation results

In order to validate the improved ro ustness of the L proposal Figure 3( ), was implemented a NO gate with three methods: standard CMOS, MRF [3] and L. he results shown in a le I are the result of SPICE simulations, using 90nm technology device models with a V_{DD} of 0.5V at temperature of 100°C. hermal noise and flicker noise was generated in each transistor, as well as, shot noise in the power supply at each time step. All kind of noise generated were amplified 100 times respect to intrinsic noise of the 90nm technology, in order to emulate the ehaviour of future technology with very noisy components.

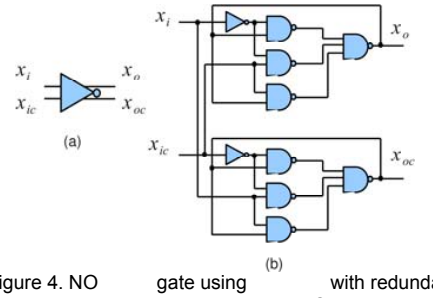


Figure 4. NO gate using with redundancy of one as well as complementary data. (a) Sym ol, and ( ) Schematic view.

As relia ility parameter to measure and compare the noise error ro ustness of these approaches is used the spurious signal (SS) factor [5], o tained etter result than previous approaches as is shown in the SS column of a le I.

Logic		Port Redun.	# Gates	SS	
				x_o	x_{oc}
NO	CMOS	0	1	8298	N. A.
	MRF	1	8	1580	2850
	tle L.	1	10	71	73

ehavior a le I. Comparative CMOS, MRF and urtle Logic techniques for a NO gate, using alternating 5,000 1's and 0's, with a period of 4ns.

VI. Conclusions

he relia ility of a circuit might e improvement respect to the increment of redundancy of ports com ined with urtle Logic (L) methodology. L has higher overhead that conventional and MRF [2] approaches, ut much etter relia ility. It requires additional settling time and may e slower than other techniques.

VII. Acknowledgments

his research work has een supported y the Spanish Ministry of Science and Innovation (MICINN) through projects EC2008-01856 and PLE20090024 (ENIAC MODERN project), with the participation of FEDER funds and "Plan E" funds, and Government of Mexico under grant 164013 CONACy and UA LX-235 PROMEP. group of research is considered a consolidated group y the MICINN.

VIII. References

[1] International echnology Roadmap for Semiconductors, online. Availa le http://pu lic.itrs.net.

[2] J. Von Neumann, "Pro a ilistic logics the Synthesis of Relia le Organisms from Unrelia le Components", Automata Studies, C. E. Shannon and J. McCarthy, eds., Princeton Univ. Press, 1956, pp. 43-98.

[3] . Nepal, R. I. Bahar, J. Mundy, W. R. Patterson, and A. Zaslavsky, "Designing logic circuits for pro a ilistic computation in the presence of noise," DAC 2005, June 13-17, 2005, Anaheim, California, USA.

[4] . Garcia, A. Calomarde, F. Moll and A. Ru io "New redundant logic function design method for extremely high noise and low-voltage scenarios", DCIS2009, Zaragoza, Spain.

[5] F Moll and A. Ru io, "Spurious signals in digital CMOS VLSI circuits: a propagation analysis," Circuits and Systems II: Analog and Digital Signal Processing, IEEE ransactions on, vol. 39, no. 10, pp. 749–752, Oct 1992.

A new probabilistic design methodology of nanoscale digital circuits

Lancelot Garcia-Leyva
Universidad Autónoma de Tlaxcala
Facultad de Ciencias Básicas, Ingeniería y Tecnología.
C/Apizaquito s/n, Apizaco, 90300
Tlaxcala, México.
Email: lancelot@eel.upc.edu

Antonio Calomarde, Francesc Moll and Antonio Rubio
Universitat Politecnica de Catalunya
Department of Electronic Engineering
C/Jordi Girona, 31, Cp. 08034.
Barcelona, Spain.
{antonio.calomarde, francesc.moll, antonio.rubio}@upc.edu

***Abstract*— The continuing trends of device scaling and increase in complexity towards terascale system on chip level of integration are putting growing difficulties into several areas of design. The intrinsic variability problem is aggravated by variations caused by the difficulties of controlling Critical Dimension (CD) in nanometer technologies. The effect of variability is the difficulty in predicting and designing circuits with precise device and circuit characteristics. In this paper, a new logic design probabilistic methodology oriented to emerging and beyond CMOS in new technologies is presented, to improve tolerance to errors due to noise, defects or manufacturability errors in single gates, logic blocks or functional units. The methodology is based on the coherence of the input redundant ports using Port Redundancy (PR) and complementary redundant ports. Simulations show an excellent performance of our approach in the presence of large random noise at the inputs.**

I. Introduction

The continuous downscaling of CMOS transistors following Moores Law has enabled the design of every time more complex electronic products and this improvement has had a great impact on the electronic industry development.

Therefore, as dimensions in devices approach physical limits of semiconductor operation they are more susceptible to several problems like process variability, susceptibility to noise, random behavior and defects. Hence, devices in ultimately scaled technologies will be increasingly unreliable. This unreliability is expected to even increase as new technologies replace todays CMOS circuits (quantum devices, carbon nanotubes, etc.) [1]. Moreover, it is clear that designers will aggressively scale down supply voltage to reduce dynamic power dissipation, the resulting reduction of logic levels approaching the thermal noise limit, and consequently reduced noise margins will expose computation to higher soft-error rates.

In order to continue building reliable circuits with these unreliable components, new design techniques must be introduced. The problem of designing reliable systems with unreliable components traces back to Von Neumann, who proposed several techniques based on N-tuple Modular Redundancy (NMR) [2] where each gate or block is replicated N times and the final output is obtained from a majority voting circuit. Other early approaches to the problem based in NMR is the case of Triple Modular Redundancy (TMR) [3], where the modules can be single gates, logic blocks, or functional units, depending on the amount of error tolerance required by the system. The system provides relief from an error caused in a single unit, but an error in two or more of the three modules will cause the logic to fail. These methods increase system reliability in the presence of transient faults. However, the main assumption underlying these redundancy processes is that the final majority voting gate is perfect and free of faults.

New proposals have appeared in the literature addressing the problem from the point of view of noise tolerance instead of exclusively considering hard defects. Among the new proposals, there is the so called Probabilistic Logic based on Markov Random Field theory (MRF) [4] which identifies valid and invalid states, and derives a logic implementation trying to reinforce valid states in order to increase their probability and decrease the probability of erroneous states. The newest approach is PCMOS, which the principal objective is developed and comprehensively characterizes probabilistic CMOS circuits (PCMOS) [5], which can be used to build energy efficient computing platforms.

Our proposal is based on the scenario that all components can fail. In order to increase circuit reliability, we propose to increase the number of each modules input and output ports, which is a form of data redundancy. This new proposal is different than the Von Neumann NMR approach in that each module is not replicated n times, but instead, the module is redesigned with I/O ports replicated n times.

The structure of the paper is as follows: in Section II, the basic concepts and principles of Probabilistic Logic are introduced. In Section III, port redundancy is introduced In Section IV, Our design proposal is introduced, In Section V, our proposal is described, explaining the implementation of an inverter as example, presenting simulation results of the different approaches with a 90nm CMOS models and measurement results after implementation with discrete CMOS components. Finally, the conclusions are presented in Section IV.

II. Probabilistic formulation

A noisy digital node does not have just two voltage values, but it presents a statistical distribution of voltages that can be characterized by a random variable with defined Probability Distribution Function (PDF). A usually employed Gaussian

pdf of a random variable x is determined by mean μ and standard deviation σ as defined in the equation (1) [6]:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} exp^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (1)$$

The probability that node voltage described by the random variable x takes a value less or equal than a given voltage V_x, is calculated by the Cumulative Probability Density Function (CPDF) in equation (2) [6].

$$P\{x \leq V_x\} = \int_{-\infty}^{V_x} p(x)dx = \int_{-\infty}^{V_x} \frac{1}{\sigma\sqrt{2\pi}} exp^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (2)$$

A. Error caused by noise in one digital node

In a noisy node, each nominal logic value (0V or 1V for example) will correspond to a random voltage obeying a pdf centered in the nominal value, as shown in Fig. 1.

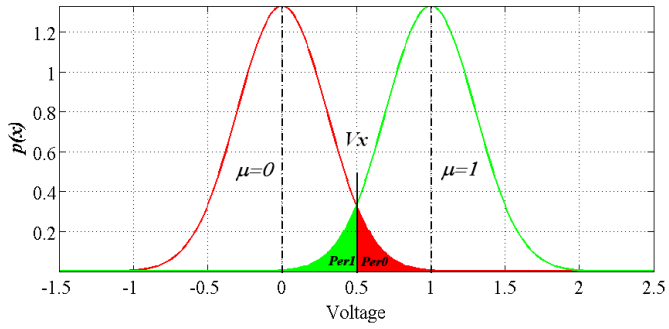


Fig. 1. The digital values of 0V and 1V voltages in a digital node under influence of Gaussian noise distributions [7].

The behavior of a noise coupled in a node is shown in Fig. 1, where the area P_{er0} and P_{er1} are the CPDF when a 0 logic is being treated as 1 logic and a 1logicis treated as 0 logic, respectively. The intersection of the PDF's for $\mu = 0$ and $\mu = 1$ occur in V_x that ideally is around $\frac{V_{DD}}{2}$ for balanced logic.

Assuming that the variance for the high and low voltage PDF's is the same, the combined probability that the data being incorrect in a node is

$$P_e = \frac{P_{er0} + P_{er1}}{2} \quad (3)$$

The probability that the data being correct in a node is

$$P_c = 1 - P_e = 1 - \frac{P_{er0} + P_{er1}}{2} \quad (4)$$

III. Port Redundancy

As an example to illustrate the proposal, let us consider the case of an inverting gate, which has only one input and one output ports. In order to improve the robustness of digital circuits under noisy scenarios, we increase the number of input and output ports. The underlying assumption is that each redundant port is generated by a different driver, so that the noise in it can be considered statistically independent.

When we increase to two the numbers of ports in an inverter gate, there are two possible ways to do it: either each redundant port takes the same value, or it takes complementary values, Fig. 2.

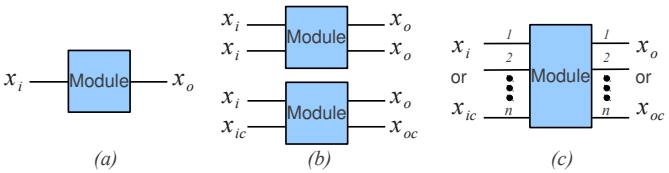


Fig. 2. Redundancy Module, a) Module without redundancy or redundancy 0, b) Data redundancy of one. The input node x can be replicating itself o its data complementary. c) Generalization for a redundancy n [7].

Independently of the replicated ports are complementary or not, the system has three possible error scenarios.

1) The values of both replicated input ports are correct P_{cmi}, Fig. 3a). The probability of this case is:

$$P_{cmi} = (1 - P_e) \cdot (1 - P_e) = (1 - P_e)^2 \quad (5)$$

2) One input port has an incorrect value P_{dmi}. This scenario is shown in Fig. 3b).

$$P_{dmi} = 2 \cdot P_e \cdot (1 - P_e) \quad (6)$$

3) The values of both input ports are incorrect P_{emi}, Fig. 3c).

$$P_{emi} = P_e \cdot P_e = P_e^2 \quad (7)$$

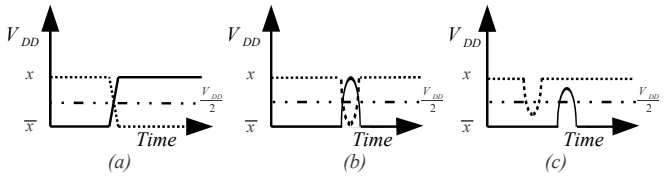


Fig. 3. Scenarios for a node in a circuit digital under high noise with redundancy of two, a) Both data are correct, b) At least one of both data is incorrect and c) Both data are incorrect [7].

The general case when the input and output ports are replicated n times, the corresponding cases have probabilities given by:

1) All replicated ports have correct values:

$$P_{cmi} = (1 - P_e)^n \quad (8)$$

2) One input port has an incorrect value P_{dmi}. This scenario is shown in Figure b).

$$P_{dmi} = \sum_{k=1}^{n-1} \frac{n!}{k!(n-k)!} (1 - P_e)^{n-k} P_e^k \quad (9)$$

3) The values of both input ports are incorrect P_{emi}, Figure c).

$$P_{emi} = P_e^n \quad (10)$$

The sum of probabilities of the three cases is one, as it should be:

$$P_{tmi} = \sum_{k=0}^{n} \frac{n!}{k!(n-k)!}(1-P_e)^{n-k}P_e^k = 1 \qquad (11)$$

A. Analysis of error reduction with port redundancy

The error probability P_{emi} depends exponentially on the number of replicated ports as can be seen in the equation (10). Figure shows the results of error probability for an inverter with redundancies from 0 to 4, for a SNR from 1 to 20 dB. As expected, if the number of replicate ports is increased, the error probability is reduced. For the specific case of redundancy one with SNR=1dB, the decrease in error probability is of 20.44%. When further increasing the port redundancy, the corresponding decrease in error probability is smaller each time: 8.26%, 5.9%, 1.68%, 0.49%, so that port redundancy beyond 4 gives negligible gains in error probability. For SNR greater than 1dB, the advantage of redundancy one over redundancy two is much lower. On the other hand, the cost in hardware of port redundancy is very high. Therefore, a redundancy of one (increasing in one the number of input and output ports) is considered in this work.

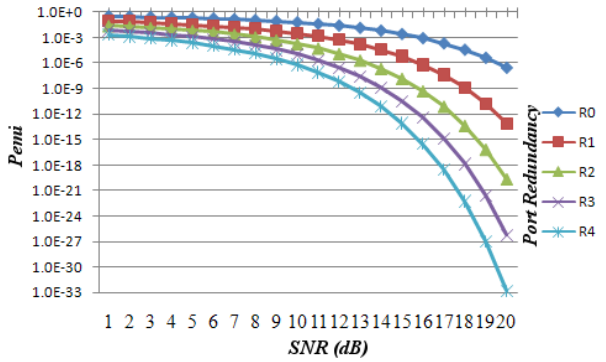


Fig. 4. P_{emi} for an inverter gate with 0,1,2,3, and 4 port redundancies.

IV. New Design Proposal for Robust Digital Circuits

The design of a given logic gate, block or function with port redundancy one and complementary redundant ports is based on the coherence of the input redundant ports. In the case of correct port values, both redundant ports should have different values. Therefore, the system identifies the case of identical redundant input ports as incorrect. In this case, the output ports of the system are forced to keep the value of the previous correct inputs. When redundant ports are different, they are considered correct by the system and the outputs take the appropriate values according to the desired logic function.

It is worth noting that there exists a probability that both redundant inputs take simultaneously an incorrect value because of noise, thus yielding an incorrect output. However, the probability of this simultaneous event is very low when there is a statistical independence of the noise in redundant ports.

TABLE I
Comparative CMOS, MRF and PR1 techniques for an inverter with redundancy of 1.

Logic		# Inputs	# Outputs	# Transistors	Overhead	SNR of input (dB)	SNR of output (dB)
NOT	Static	1	1	2	1.0	1.0	4.51
	MRF	2	2	20	10.0	1.0	18.44
	PR1	2	2	48	24.0	1.0	42.55

V. Inverter Gate: PR1

Based on the previous section an inverter is designed with redundancy one and complementary redundant ports. We call this approach PR1. The resulting circuit is presented in Fig. 5, where x_{ic} and x_i are the complementary input signals and x_{oc} and x_o correspond to complementary output signals.

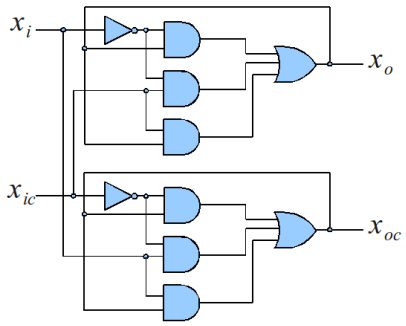


Fig. 5. Schematic view of an inverter using PR1.

A. Simulation results

The results shown in Table I correspond to SPICE simulations of three CMOS implementations of the inverter function using 90nm technology device models and a V_{DD} of 0.5V at room temperature. We implemented the inverter function using standard CMOS, MRF [4], and our approach shown in the schematic in Fig. 5. The input signal was generated by adding white Gaussian noise with 0V mean and standard deviation of 0.6V to both logic levels 1 (0.5V) and 0 (0V). Uncorrelated noise sources are applied to each of the inputs, while the outputs are left unforced.

B. Physical implementation

Conventional CMOS, MRF and PR1 inverters have been implemented with discrete gate circuits using the standard CMOS gates mounted in a PCB, Fig. 6, and it has been tested under the following conditions: Power supply V_{DD}=3.3V, input voltage in x_{ic}=3.3V, input voltage in x_i additive white Gaussian noise with mean 0V and deviation standard of 0.8 V. Both waveforms x_{ic} and x_i are shown in Fig. 7a). The noise at the inputs causes the standard CMOS output x_o CMOS to become very noisy and potentially erroneous as shown in Fig. 7a). The MRF inverter output x_{oc} MRF is also erroneous as

shown in Fig. 7b). However, the PR1 (x_o our and x_{oc} our) implementation gives clean correct outputs as shown also in Fig. 7b).

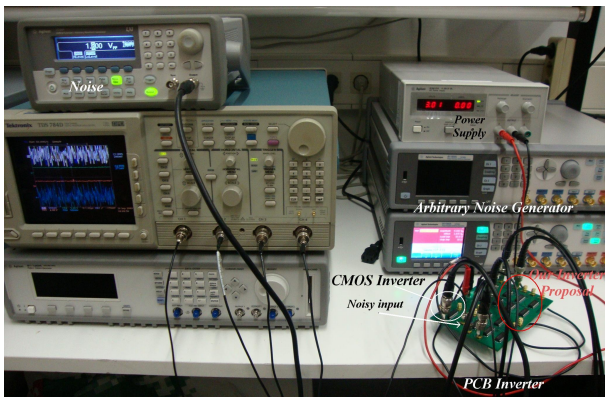


Fig. 6. Physical implementation with discrete gates.

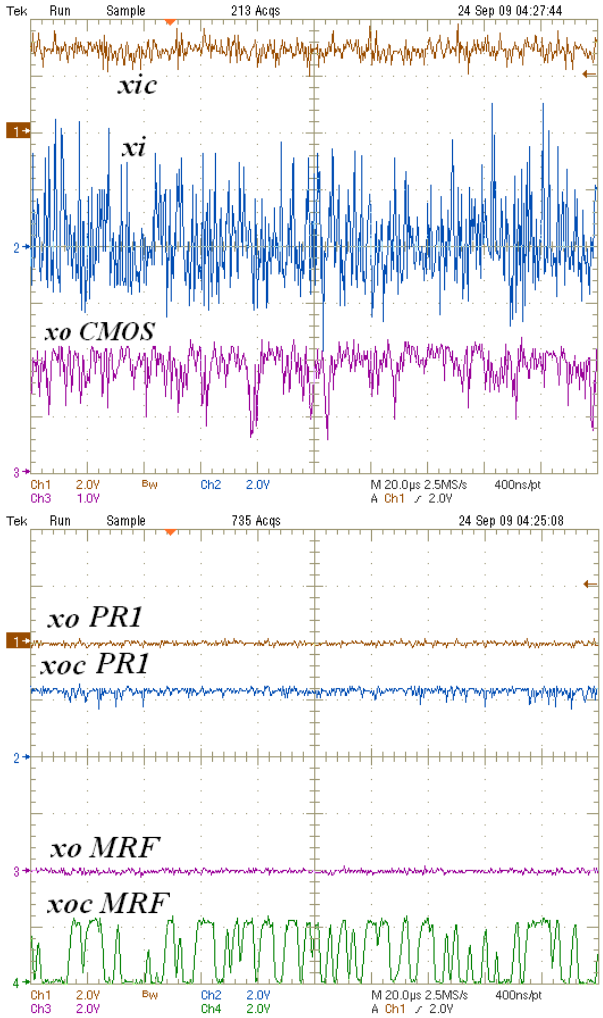


Fig. 7. Laboratory measures for the three types of inverter with noisy input. Conventional CMOS in (a) and PR1 and MRF in (b).

VI. Conclusions

This paper shows how increasing port redundancy decreases the error probability. This result is the basis for a new way of designing logic blocks which are tolerant to a high level of noise (SNR on the order of 1dB), under the assumption of statistically independence of noise in the redundant ports.

The simulation and experimental results show a much better noise tolerance of our approach compared to other approaches, at the cost of a considerable overhead in hardware complexity. This level of overhead will be plausible in future beyond CMOS deep nanoscale technologies where the scenario of low SNR considered in this paper and the design strategy we propose will be adequate.

Acknowledgment

This research work has been supported by the Spanish Ministry of Science and Innovation (MICINN) through projects TEC2008-01856 and PLE2009-0024 (ENIAC MODERN project), with the participation of FEDER founds and "Plan E" funds, also by the project TRAMS of the FP7 of the European Commission 248789, as well as Government of Mexico under grants 164013 CONACyT and UATLX-235 PROMEP. The group of research is considered a consolidated group by the MICINN.

References

[1] International Technology Roadmap for Semiconductors, 2009 Edition executive summary, online available in *http://public.itrs.net*.

[2] J. Von Neumann, *"Probabilistic logics the Synthesis of Reliable Organisms from Unreliable Components*, Automata Studies, C. E. Shannon and J. McCarthy, eds., Princeton Univ. Press, 1956, pp. 43-98.

[3] W. H. Pierce, *"Failure-Tolerant computer design*, Academic Press, 1965.

[4] K. Nepal, R. I. Bahar, J. Mundy, W. R. Patterson, and A. Zaslavsky, *"Designing logic circuits for probabilistic computation in the presence of noise*, DAC 2005, June 13-17, 2005, Anaheim, California, USA.

[5] Pinar Korkmaz, *"Probabilistic CMOS (PCMOS) in the Nanoelectro-nics Regime"*, PhD thesis, Georgia Institute of Technology, Dec. 2007.

[6] Athanasios Papoulis and S. Unnikrishna Pillai, *"Probability, Random Variables and Stochastic Processes*, Fourth Edition. New York, year 2002.

[7] Garcia-Leyva, L.; Calomarde, A.; Moll, F.; Rubio, A., *"Turtle Logic: A new probabilistic design methodology of nanoscale digital circuits*, Circuits and Systems (MWSCAS), 53rd IEEE International Midwest Symposium on. Seattle, Washington, August 1st-4th, 2010. ISSN: 1548-3746.

Microelectronics Journal 42 (2011) 1359–1369

Contents lists available at SciVerse ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

Review

New redundant logic design concept for high noise and low voltage scenarios

Lancelot García-Leyva [a,b,n], Dennis Andrade [b], Sergio Gómez [b], Antonio Calomarde [b], Francesc Moll [b], Antonio Rubio [b]

[a] Universidad Autónoma de Tlaxcala, FCBIyT, C/Apizaquito s/n, Apizaco, 90300 Tlaxcala, Mexico
[b] Universitat Politènica de Catalunya, Department of Electronic Engineering, C/Jordi Girona 31, 08034 Barcelona, Spain

a r t i c l e i n f o

Article history:
Received 14 April 2011
Received in revised form
13 September 2011
Accepted 16 September 2011
Available online 7 October 2011

Keywords:
Redundancy
Fault tolerance
High noise
Robustness
Low voltage
Error probability

a b s t r a c t

This paper presents a new redundant logic design concept named Turtle Logic (TL). It is a new probabilistic logic method based on port redundancy and complementary data, oriented toward emerging technologies beyond CMOS, where the thermal noise could be predominant and the reliability of the future circuits could be limited. The TL is a technology independent method, which aims to improve error tolerance when these errors are caused by noise within logic and functional units, sequential elements, and in general synchronous pipeline Finite State Machines. Turtle Logic operation is based on the consistency relation of redundant inputs. In the case of discrepancy, the output of the system keeps the previous value, therefore avoiding the propagation of incorrect inputs. A two's complement 8 x 8-bit pipelined Baugh–Wooley multiplier is implemented, on which several experiments reveal a perfect tolerance (0% errors) to single line discrepancies for both primary and internal nodes, with a cost of lost clock periods between 6% and 25%. The error ratio for the proposed Turtle Logic implementation with double discrepancies in both true and complementary lines are lower than 0.1% when the noise affects primary input nodes, and lower than 0.9% when the noise affects internal nodes.

[n] Corresponding author at: Universitat Politènica de Catalunya, Department of Electronic Engineering, C/Jordi Girona 31, 08034 Barcelona, Spain. Tel.: þ34 934016766; fax: þ34 934016756.
E-mail addresses: lancelot.garcia@upc.edu (L. García-Leyva), antonio.calomarde@upc.edu (A. Calomarde), francesc.moll@upc.edu (F. Moll), antonio.rubio@upc.edu (A. Rubio).

0026-2692/$ - see front matter & 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mejo.2011.09.007

Novel redundant logic design for noisy low voltage scenarios

Lancelot Garcia-Leyva
Universidad Autónoma de Tlaxcala
Facultad de Ciencias Básicas, Ingeniería y Tecnología.
C/Apizaquito s/n, Apizaco, 90300
Tlaxcala, Mexico.
Email: lancelot.garcia@upc.edu

Antonio Calomarde, Francesc Moll and Antonio Rubio
Universitat Politecnica de Catalunya
Department of Electronic Engineering
C/Jordi Girona, 31, 08034, Barcelona, Spain.
{antonio.calomarde, francesc.moll, antonio.rubio}@upc.edu

***Abstract*— The concept worked in this paper named Turtle Logic (TL) is a probabilistic logic method based on port redun- dancy and complementary data, oriented to emerging CMOS technologies and beyond, where the thermal noise could be predominant and the reliability of the future circuits will be limited. The TL is a technology independent method, which aims to improve tolerance to errors due to noise in single gates, logic and functional units. The TL operation is based on the consistency relation of redundant inputs. In case of discrepancy; the output of the system keeps the previous value. Therefore, it avoids the propagation of incorrect inputs. Simulations show an excellent performance of TL in the presence of large random noise at the inputs with a practical full tolerance to input with a signal to noise ratio of 5dB. Turtle Logic in accordance to Kullback Leibler Distance noise immunity measurement for a NOT gate is approximately 3.6X, 13.4X and 20.9X times better than MRF, DCVS and standard CMOS techniques, respectively, when the gates are operate, with a power supply of 0.15 volts, a temperature of 100 ^{o}C and noisy inputs with Additive White Gaussian Noise with zero mean and a standard deviation of 60mV.**

978-1-4673-4900-0/13/$31.00 ©2013 IEEE