CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

Unidad Irapuato

Implementación de un hardware reconfigurable para la simulación dinámica de redes de regulación genética mediante la búsqueda exhaustiva de atractores

Tesis que presenta

Biol. Carlos Andrés Hernández Martínez

Para obtener el grado de

MAESTRO EN CIENCIAS

En la especialidad de

BIOTECNOLOGÍA DE PLANTAS

Director de tesis:

Dr. Agustino Martínez Antonio

Asesores:

Dra. Selene Lizbeth Fernández Valverde

Dr. Octavio Martínez de la Vega

Dr. Ismael Sánchez Osorio (invitado)

Irapuato, Guanajuato Agosto, 2017

Agradecimientos

Al CONACYT por la beca de maestría otorgada.

A mi asesor el Doctor Agustino Martínez Antonio por darme la oportunidad de trabajar en su grupo de investigación, gracias también por sus consejos y asesoría brindados para la realización de este trabajo.

A mis asesores, el Dr. Octavio Martínez de la Vega, la Dra. Selene Lizbeth Fernández Valverde y el Dr. Ismael Osorio Sánchez por sus sugerencias y aportaciones y disponibilidad durante la realización de este proyecto.

A mis compañeros de laboratorio por su grata compañía y consejos.

A mi familia, por todo su cariño apoyo y confianza.

A todo el personal de CINVESTAV Irapuato por las facilidades otorgadas durante la realización de esta maestría

Contenido

Resumen

Durante las dos últimas décadas, la generación y anotación masiva de funciones genéticas e interacciones regulador-promotor han resultado en la determinación extensiva de diagramas de interconexión de GRNs (*Gene Regulatory Networks)* para varios organismos. Este conocimiento ha contribuido significativamente a la caracterización de los principios de organización globales detrás de la estructura de las GRNs, pero es insuficiente *per se* para obtener un entendimiento predictivo de cómo la información genética se traduce en fenotipos. Para aspirar a tal entendimiento se necesita indagar en las propiedades emergentes de los circuitos genéticos, que surgen como producto de interacciones variantes en el tiempo entre sus componentes. La interrogación experimental *in vivo* de la dinámica de las redes a escala celular aún no es posible con la tecnología actual. Además, aún si se pudiera acceder a este tipo de información, seguiría siendo un desafío dar sentido a la enorme cantidad de observaciones con resolución temporal resultantes. Por lo tanto, en lugar de facilitar la interpretación y entendimiento mecanístico de las relaciones entre genotipo y fenotipo, la generación de datos pudiera resultar confusa en ausencia de un marco conceptual que motive experimentos particulares. En la era actual de la biología de sistemas, dicho marco conceptual es provisto por modelos matemáticos y computacionales.

A pesar de su complejidad biológica, las GRNs se pueden conceptualizar en el lenguaje de la teoría de grafos en términos de nodos, que representan genes, y en aristas que indican interacciones regulatorias. Hay un amplio rango de enfoques de modelación para describir estas funciones regulatorias, que van desde representaciones Booleanas de los estados de los nodos, hasta descripciones de sus dinámicas de manera estocástica y con resolución espacial

El formalismo matemático más simple para explorar la dinámica de una red de regulación son los modelos Booleanos. Estos han sido utilizados históricamente como una primera aproximación para describir las interacciones entre los genes que conforman una red de regulación. El conjunto de todas las funciones Booleanas para los genes de una red puede ser visto como un sistema de ecuaciones que describe la dinámica del estado de la red. Mediante este sistema de ecuaciones, se pueden calcular las transiciones en el estado global de la red a partir de una condición inicial definida. El objetivo de estas simulaciones es encontrar patrones recurrentes en el comportamiento de la expresión genética llamados atractores.

Sin embargo, aún en un paradigma Booleano, la simulación de la dinámica de una red regulatoria representa un reto computacional, debido a que, por la propia naturaleza del problema, el tiempo de cómputo para encontrar todos los atractores en la dinámica de una red crece exponencialmente con el número de genes en la red.

Las herramientas de software actuales permiten al usuario realizar la búsqueda exhaustiva de atractores en redes de regulación pequeñas con relativa simplicidad y tienen cierta flexibilidad respecto al uso de algoritmos heurísticos para abordar redes de mayor tamaño. Sin embargo, resulta relevante enfatizar que la búsqueda exhaustiva de atractores en redes de regulación no reducidas de tamaño mayor a 20 genes se encuentra sujeta al desarrollo de nuevos paradigmas algorítmicos y de plataformas computacionales de alto desempeño.

En este trabajo, describimos nuestra contribución en este último aspecto. En particular, abordamos la búsqueda exhaustiva de atractores en redes Booleanas desde un paradigma de cómputo basado en una implementación algorítmica directa en hardware usando dispositivos

conocidos como FPGAs (*Field Programmable Gate Array*). Específicamente, se diseñó e implementó una arquitectura de hardware para la búsqueda de atractores en la dinámica de redes de regulación modeladas bajo un paradigma Booleano. Esta arquitectura se validó mediante la simulación dinámica de dos redes de regulación reportadas en la literatura.

Para la simulación de la dinámica correspondiente a una red de 21 genes, la implementación en hardware desarrollada en este trabajo superó el desempeño de implementaciones de software publicadas en la literatura, tanto basadas en exploración exhaustiva como en métodos heurísticos.

En comparación con otras implementaciones en hardware reportadas, nuestro diseño permite al usuario interactuar con la capa de hardware desde una plataforma de software, y reporta a mayor detalle los resultados de las simulaciones.

Abstract

In the last two decades, the massive generation and annotation of gene functions and regulators-DNA interactions have resulted in the extensive determination of interconnection diagrams for GRNs of many organisms. This knowledge has contributed significantly to the characterization of the global organization principles behind the structure of GRNs, but this knowledge itself is not enough to obtain a predictive understanding on how genetic information is translated into phenotypes. To aspire to such understanding, we need to study the emergent properties of genetic circuits, which arise from the time-varying interactions among their components.

The in vivo experimental interrogation of the dynamics of regulatory networks at a cellular scale is currently not feasible. Besides, even if it were possible to attain such information, it would still be a challenge to make sense of the enormous amount of time-resolved observations. Thus, instead of simplifying the interpretation and mechanistically understanding of the relationship between genotype and phenotype, the generation of such data may turn confusing in the absence of a conceptual framework that motivates experiments. In the current era of systems biology, such a conceptual framework is given by mathematical and computational models. Despite their complexity, GRNs can be conceptualized in the language of graph theory in terms of nodes that represent genes, and edges that represent regulatory interactions. There is a wide spectrum of mathematical modelling approaches to describe such regulatory functions, ranging from Boolean representations of the node states, to descriptions of their dynamics that incorporate stochasticity and spatial resolution.

The simplest mathematical formalism to explore the dynamics of a regulatory networks are Boolean models. These models have been historically used as a first approximation to describe the interactions among genes which conform the regulation network. The set of all the Boolean functions for the genes in a network can be considered as a system of equations that describe the dynamics of the network state. Through this system of equations, it's possible to calculate the transitions in the global state of the network for a given initial condition. The goal of such simulations is to find recurrent patterns in the behavior of gene expression, which are called attractors.

However, even in a Boolean paradigm, the simulation of a network's dynamics represents a computational challenge because, by the very nature of the problem, the computational time to find all attractors grows exponentially with the number of genes in the network

Current software tools, allow the user to easily perform exhaustive attractor search in small regulatory networks, and have some flexibility in the use of heuristic methods to study larger networks. However, it is relevant to point out that the exhaustive search of attractors in non-reduced regulatory networks is subject to the development of new algorithmic paradigms, and of high-performance computing platforms. The contribution of the current Work falls in the former aspect of technological development. Particularly, we tackle the exhaustive search of attractors in Boolean networks from a computational paradigm based in the direct implementation of an algorithm in hardware using devices known as FPGAs. Specifically, we designed and implemented a hardware architecture for the search of attractors in the dynamics of regulatory networks under a Boolean paradigm. This architecture was validated by simulating the dynamics of two regulatory networks reported in the literature.

For the simulation of the dynamics corresponding to a 21-gene network, the hardware implementation developed in this work surpassed the performance of software implementations published in the literature, based on exhaustive search as well as on heuristic methods.

In contrast with other reported hardware implementations, our design allows the user to interact with the hardware layer from software, and reports the results of the simulation more comprehensively.

Índice de figuras

Índice de tablas

Índice de ecuaciones

Abreviaturas

ARM: *Advanced Reduced-instruction-set-computer Machine,* Máquina de RISC Avanzada

CAC: *Colitis-Associated Colon Cancer*, Cáncer de Colon Asociado a Colitis

DTC: *Device Tree Compiler*, Compilador de Árboles de Dispositivos.

EDA: *Electronic Design Automation,* Automatización de Diseño Electrónico

FPGAs: *Field Programmable Gate Arrays,* Arreglos de Compuertas Programables en Campo

FSM: *Finite State Machine,* Máquina de Estados Finitos

GRN: *Gene Regulatory Network,* Red de Regulación Genética

HDLs: *Hardware Description Languages*, Lenguajes de Descripción de Hardware

NP: *Nondeterministic Polynomial*, Polinomial No-determinista

PAD: *Partition-based Attractor Detection tool*, Herramienta de Detección de Atractores basada en Particiones

ROM: *Read Only Memory*, Memoria de Solo Lectura

SAT: Satisfactibilidad Booleana

SoC: *System on a Chip*, Sistema en un Chip

SPL: *Secondary Program Loader*, Programa Cargador Secundario

TCL: *Tool Command Language*, Lenguaje de Herramientas de Comando

TF: *Transcription Factor*, Factor de Transcripción

VHDL: *Very-High-Speed-Integrated-Circuit HDL*, Lenguaje de Descripción de Hardware para Circuitos Integrados de Muy Alta Velocidad

Glosario

Grafo: Conjunto de elementos llamados nodos unidos por enlaces llamados aristas. Las aristas permiten representar relaciones entre los elementos del conjunto.

Circuito electrónico: Red eléctrica que contiene una trayectoria cerrada.

Circuito integrado: Conjunto de circuitos electrónicos incorporados en una sola pieza de material semiconductor.

Kernel: Programa que constituye la parte fundamental del sistema operativo. Su función principal es gestionar el acceso de los programas que se ejecutan en una computadora a los recursos de hardware.

Módulo: Elemento con función propia, que se integra con otros elementos para formar un sistema mayor.

Onda cuadrada: Onda cuyo valor alterna a intervalos regulares entre dos valores extremo de manera casi instantánea.

Bare metal: También llamado "*bare machine*" es una expresión que se usa para referirse a la ejecución de un programa en un procesador sin la intervención de un sistema operativo.

Memoria: Dispositivo que almacena datos, ya sea de manera temporal o permanente.

Memoria Flash: Medio de almacenamiento de datos que permite guardar información sin necesidad de corriente. La información almacenada puede ser borrada o sobrescrita mediante señales eléctricas.

ROM: Medio de almacenamiento que permite la lectura de la información, pero no su escritura o eliminación.

Reloj: Desde un punto de vista electrónico, es un dispositivo que oscila a intervalos regulares generando una señal que se convierte en una secuencia alternante de unos y ceros. Esta oscilación por lo general se obtiene mediante un cristal de cuarzo produce una señal que oscila a una frecuencia precisa. Esta señal es procesada por un circuito eléctrico que la convierte en una señal de onda cuadrada, la cual es usada para actualizar el estado de los componentes en un circuito electrónico.

Frecuencia de Reloj: Indica el número de veces en un intervalo de tiempo que oscila la onda cuadrada generada por un reloj. Esta frecuencia determina qué tan rápido puede actualizarse el estado de un circuito electrónico, y por lo tanto el número de instrucciones que puede realizar en un intervalo de tiempo.

Latch: Circuito electrónico que puede mantenerse en uno de dos estados posibles. Los estados del circuito corresponden a los valores de "1" y "0" del bit almacenado. El circuito cambia de estado cuando cambian los valores de sus señales de entrada.

Flip-flop: Circuito electrónico que puede mantenerse en uno de dos estados posibles. Los estados del circuito corresponden a los valores de "1" y "0" del bit almacenado. El circuito actualiza su estado solamente en el flanco de subida de una señal de reloj. Fuera del flanco de subida, el circuito mantiene su estado independientemente del valor de sus señales de entrada.

I Introducción

Dentro de las redes de regulación genéticas (GRNs por sus siglas en inglés) existen factores de transcripción (TFs, por sus siglas en inglés) referidos como reguladores globales (Gottesman, 1984), que pueden llegar a regular cientos de genes, ya sea de manera directa o indirecta mediante la regulación de otros TFs, y por lo tanto presentan un efecto pleiotrópico sobre el fenotipo celular (Martínez-Antonio y Collado, 2003). La presencia de este tipo de genes indica que las GRNs operan bajo una estructura jerárquica; aunque en muchos casos los reguladores globales también actúan en conjunto con reguladores locales para mediar el control de la expresión de sus genes regulados. Estando inmersos los reguladores globales en lazos de retroalimentación y otras interacciones regulatorias complejas, las GRNs no se pueden conceptualizar únicamente dentro de una estructura jerárquica. Considerando estas observaciones, se puede decir que las respuestas fenotípicas mediadas por TFs son resultado de contribuciones de una regulación local y otra global (Martínez-Antonio y Collado, 2003). Dentro de esta conceptualización, es posible identificar sub-circuitos inmersos en las GRNs, que pueden ser vistos como módulos que controlan aspectos particulares de los fenotipos celulares y cuyo comportamiento es una función del conjunto de los TFs presentes en la célula en un momento determinado (Erwin y Davidson, 2009).

Aunque los primeros estudios acerca de circuitos regulatorios se realizaron originalmente en bacterias (Jacob y Monod, 1961; Yaniv, 2011), se han encontrado resultados análogos en eucariotas, en donde se han identificado circuitos regulatorios involucrados en desarrollo y diferenciación celular (Davidson *et al.,* 2002). Esto señala la existencia de principios de organización generales en las redes que se manifiestan a través de la recurrencia de elementos topológicos dentro de las mismas (Alon,, 2007). Por lo tanto, en este trabajo hemos considerado como casos de estudio un circuito genético asociado al control del ciclo celular de la bacteria Gram negativa *Caulobacter crescentus*, así como un circuito genético involucrado en la proliferación celular en cáncer de colon asociado a colitis en humanos. Estos circuitos se describen a detalle en la sección 1.4.1 y 1.4.2 del presente trabajo.

1.1 Entendimiento dinámico de la regulación genética

Durante las dos últimas décadas, la generación masiva de datos y la anotación de funciones genéticas e interacciones TF-DNA a escala de genomas completos han resultado en la determinación extensiva de diagramas de interconexión de GRNs para varios organismos (Cheng *et al.,* 2011; Gama-Castro *et al.,* 2011; Pauling *et al.,* 2011). A pesar de que en la actualidad no se cuenta con una GRN completa para ningún organismo, el catálogo de interacciones conocidas y componentes moleculares que intervienen en estas redes está creciendo y cada vez se vuelve más detallado. Si bien este conocimiento ha contribuido significativamente a la caracterización de principios de organización globales detrás de la estructura de las GRNs, no es suficiente por sí mismo para obtener un entendimiento predictivo de cómo la información genética codificada en el DNA y los estados celulares globales se traducen en fenotipos. Se ha vuelto más claro que nunca, que para aspirar a tal entendimiento se necesita indagar en las propiedades emergentes de los circuitos genéticos, las cuales, en contraste con las que corresponden a sus componentes moleculares considerados de manera individual, surgen como producto de interacciones variantes en el tiempo (Purvis y Lahav, 2013).

Desde el punto de vista de la biología molecular, si pudiéramos determinar la función de cada una de las moléculas involucradas en un fenotipo celular particular, entonces en principio sería posible entender cómo dicho fenotipo surge de los componentes moleculares de la célula. Sin embargo, conforme se descubrieran más genes, productos génicos, y otras moléculas, seguiría siendo un reto tratar con el elevado número de interacciones resultantes para extraer conocimiento. En consecuencia, es necesario tener una forma de unificar los datos en un marco conceptual coherente, de tal manera que dichos fenómenos biológicos puedan ser estudiados y entendidos como un sistema. Esta integración también requeriría tomar en cuenta la variación temporal en la expresión de los genes involucrados en el desarrollo de los fenotipos observados, pues tal dinámica puede ser decisiva en la emergencia los mismos (Ryan y Shapiro, 2003). Por lo tanto, al dejar de enfocarnos en genes y moléculas individuales, sino en las propiedades dinámicas globales de las redes regulatorias, sería posible avanzar el entendimiento del origen de los fenotipos celulares a partir de sus componentes celulares. Esta meta puede ser facilitada mediante el uso del lenguaje formal y preciso de las matemáticas (Cohen, 2004).

1.2 Modelación matemática de la dinámica de la regulación

La interrogación experimental in vivo de la dinámica de las redes a escala celular aún no es posible con la tecnología actual. Además, aún si se pudiera acceder a este tipo de información, seguiría siendo un desafío dar sentido a la enorme cantidad de observaciones con resolución temporal resultantes. Por lo tanto, en lugar de facilitar la interpretación y entendimiento mecanístico de las relaciones entre genotipo y fenotipo, la generación de datos pudiera resultar confusa en ausencia de un marco conceptual que motive experimentos particulares. En la era actual de la biología de sistemas, dicho marco conceptual es provisto por modelos matemáticos y computacionales. El poder de la abstracción asociado con el desarrollo de modelos teóricos se ha vuelto esencial para la organización del conocimiento biológico, la generación de hipótesis acerca de los principios de la operación celular y la predicción de resultados que pueden ser confrontados con la evidencia experimental (Scott *et al.,* 2010).

A pesar de su complejidad biológica, las GRNs se pueden conceptualizar en el lenguaje de la teoría de grafos en términos de nodos, que representan genes, y en aristas que indican interacciones regulatorias, es decir, activación, represión o regulación dual por TFs. Adicionalmente, se puede incluir información dinámica acerca de cómo el estado de cada nodo en el grafo cambia a través del tiempo, dependiendo del estado de los otros nodos. Hay un amplio rango de enfoques de modelación para describir estas funciones regulatorias de "entrada-salida", que van desde representaciones Booleanas de los estados de los nodos, hasta descripciones de sus dinámicas de manera estocástica y con resolución espacial (Sánchez-Osorio *et al.,* 2014).

1.2.1 Modelos lógicos Booleanos

Dentro de la célula, las concentraciones de los TFs pueden asumir, en principio, cualquier valor dentro del dominio de los números reales positivos. A su vez, estas concentraciones pueden variar en el tiempo de manera continua. Dadas estas características, el formalismo matemático inmediato para describir estas cantidades resultan ser las ecuaciones diferenciales (Chen *et al.,* 2005; Sakamoto e Iba, 2001; Chen *et al.,* 1999). Sin embargo, estas ecuaciones tienen ciertos requerimientos de datos para su construcción, incluyendo parámetros cinéticos de producción y degradación de los TFs, así como conocimiento de los mecanismos regulatorios a nivel molecular. Para la gran mayoría de los TFs, estos parámetros se desconocen completamente y

se tienen que aproximar mediante técnicas de ajuste y optimización de los mismos, la mayoría de los cuales resultan ser poco fiables dada la escasez de mediciones experimentales con alta resolución temporal (De Jong, 2010). A la escala de una red con decenas de nodos, nuestra presente ignorancia de los parámetros ya mencionados, así como la dimensionalidad del problema asociado a la solución del sistema de ecuaciones diferenciales correspondiente, han limitado el uso de este esquema en favor de enfoques de modelación más simples para la exploración de la dinámica de las redes (Bornholdt y Rolf, 2000).

La forma más sencilla de describir la expresión de un gen es como una variable, $\sigma i\ (t)$, la cual puede ser representada usando solamente dos estados, "1" y "0", que representan "activo" e "inactivo" respectivamente (Fig. 1a). Este tipo de variables binarias se conocen como variables Booleanas, en alusión a George Boole, quien desarrollo el formalismo matemático para tratar con este tipo de valores (Boole, 1848). Los modelos Booleanos han sido utilizados históricamente como una primera aproximación para describir, al menos de forma cualitativa, las interacciones entre los genes que conforman una red de regulación (Kauffman, 1969; Thomas y Mikulecky, 1978).

En un modelo Booleano, la dinámica de la expresión de un gen se puede representar como una serie de transiciones entre valores de activo o inactivo a través de intervalos de tiempo discretos (Fig. 1b). Aunque a primera vista este tipo de modelos pueden parecer muy restringidos, es necesario tomar en cuenta que todo modelo es solamente una representación que se construye a partir de un rango limitado de datos, que se ajustan en menor o mayor medida al comportamiento biológico observado. Tomando esto en cuenta, es válido elegir un modelo por su practicidad, aun si nunca se ajusta exactamente a los datos observados.

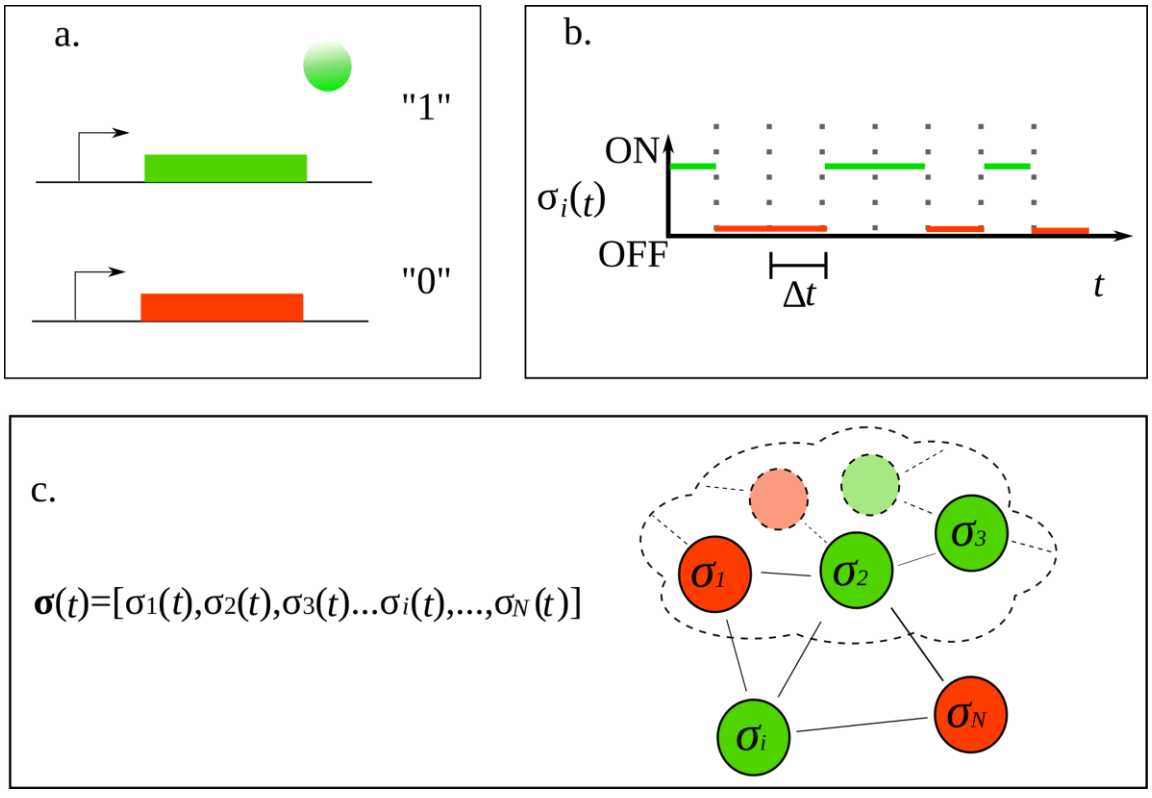

Figura 1. Expresión genética modelada desde un formalismo Booleano. a) El estado de un solo gen es representado como "1" o "0". El estado activo del gen es representado en color verde, mientras que el estado reprimido se indica en rojo. b) La dinámica de la expresión de un gen es representada a través

del tiempo. c) El estado de una red es representado como un vector. Figura tomada de Sánchez-Osorio *et al*., 2017.

Para una red regulatoria de *N* genes, es posible codificar el estado de toda la red en el tiempo *t* mediante un vector $\boldsymbol{\sigma}(t) = [\sigma_1(t), \sigma_2(t), \sigma_3(t)..., \sigma_i(t), \ldots, \sigma_N(t)]$, donde el estado del gen i es representado por la variable $\sigma_i(t)$, como se ilustra en la figura 1c. El estado de cada gen *i* dentro de la red es afectado por los *k* genes que lo regulan directamente. Por lo tanto, la dinámica de la expresión del gen i puede ser especificada como una función de los estados de los genes que lo regulan, $\sigma_{i1}(t), \ldots, \sigma_{ik}(t)$. De manera más precisa, la evolución en el tiempo del valor de σ_i puede ser escrita como $\sigma_i(t+1)=f_i(\sigma_{i1}(t), \sigma_{i2}(t), \ldots, \sigma_{ik}(t))$, donde f_i es la función Booleana que representa la regulación de un gen *i* (Fig. 2). La función de un gen *i* describe el estado de éste en el siguiente intervalo de tiempo, según el estado actual de los genes que lo regulan.

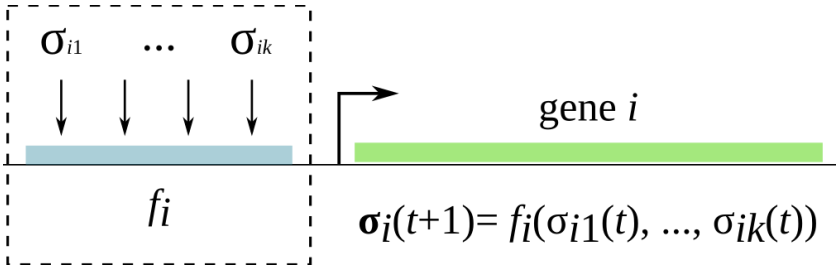

Figura 2. Dinámica del gen *i* como función de los genes que lo regulan. En esta figura, f_i representa a la función regulatoria lógica que vincula el estado de expresión del gen *i* en el tiempo t+1, con el estado en el tiempo *t* de los *k* genes que lo regulan. Figura tomada de Sánchez-Osorio *et al*., 2017.

La función Booleana para la expresión de un gen, también llamada regla lógica, se formula mediante funciones lógicas tales como ADN, OR y NOT. Tomemos como ejemplo al gen *gcrA*. Este pertenece a la red de regulación asociada al ciclo celular de *C. crescentus*, que se describe en la sección 1.3.1. Este gen se expresa en el tiempo *t*+1 (*ie.*, $\text{GcrA}^{t+1} = 1$) cuando en el tiempo *t* la proteína DnaA está presente y CtrA ausente. Esta relación puede expresarse con los operadores ADN y NOT, como se muestra en la ecuación 1.

GcrA^{t+1}= DnaA AND (NOT CtrA).

Ecuación 1. Función Booleana del gen *gcrA*.

1.2.2 Simulación de la dinámica de una red

El conjunto de todas las funciones Booleanas para los genes de una red puede ser visto como un sistema de ecuaciones que describe la dinámica del estado de la red. Mediante este sistema de ecuaciones, se pueden calcular las transiciones en el estado global de la red a partir de una condición inicial definida. Para ilustrar este concepto, tomaremos como ejemplo la red del control del ciclo celular en *C. crescentus* descrita en la sección 1.3.1. Esta red está formada por cinco nodos, enumerados como el vector **σ** = [CtrA, GcrA, DnaA, CcrM, SciP]. Asumamos que en un tiempo inicial t_0 los valores de los nodos corresponden al vector $\boldsymbol{\sigma}(t_0)$ = [0, 0, 1, 0, 1]. En cada intervalo de tiempo subsecuente, el estado de cada nodo se actualiza según su propia función Booleana, como se ilustra en la figura 3a. Las transiciones entre los estados de la red son deterministas, lo que en este contexto significa que, para cada estado de red en un

intervalo de tiempo, corresponde solamente un estado siguiente posible. Este esquema de actualización de la red en el que todos los nodos se actualizan en un intervalo de tiempo es conocido como síncrono. Existen esquemas de actualización asíncronos en los que no todos los nodos se actualizan simultáneamente (Saadatpour y Albert, 2013), pero por el resto de este trabajo nos enfocaremos solamente en el esquema síncrono.

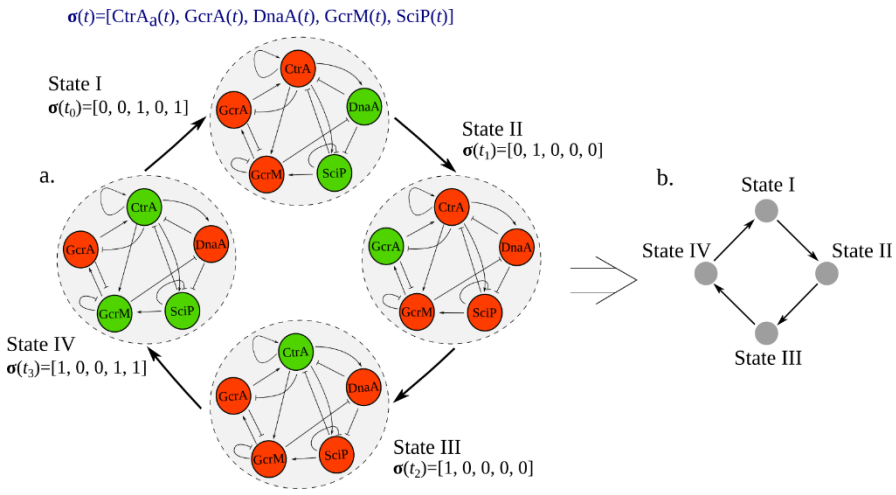

Figura 3. Transiciones de estados en una red. a) Transiciones entre estados de la red de *C. crescentus* a partir de un estado inicial $\boldsymbol{\sigma}(t_0)$ = [0, 0, 1, 0, 1]. Los nodos activos en un estado de tiempo se señalan en verde, mientras que los inactivos en rojo. b) Representación abstracta de la dinámica como un diagrama de transición de estados. Figura tomada de Sánchez-Osorio *et al*., 2017.

La representación visual de las transiciones entre los estados de la red mostrada en la figura 3a, resulta poco práctica para redes de mayor tamaño. Por este motivo, se recurre a una representación abstracta llamada diagrama de transición de estados. En esta representación la dinámica de la red se conceptualiza como una red, en la que cada nodo corresponde a un estado de la red y cada arista a una transición entre estados (Fig. 3b).

A la sucesión de estados por los que transita la red a partir de un estado inicial se le conoce como trayectoria. En la trayectoria de la dinámica presentada en la figura 3, el estado IV transita de regreso al estado I. Dado que las transiciones entre estados son deterministas, la trayectoria de la red transitará por la misma serie de estados de manera indefinida. A esta clase de comportamiento recurrente en la dinámica de una red se le conoce como atractor. El número de estados contenidos en un atractor se denomina periodo. Un atractor con periodo de más de 1 es llamado atractor cíclico, mientras que un atractor con solo un estado es llamado puntual o fijo. En principio, como las transiciones entre estados son deterministas, y hay un número finito de estados posibles, todo estado inicial en la dinámica de una red Booleana síncrona converge eventualmente en un atractor (Saadatpour y Albert, 2013).

Para encontrar todos los atractores de una red, se simula la dinámica de la red correspondiente tomando como estados iniciales a todos los estados posibles que puede adoptar la red. Al conjunto de todos los estados de red posible se le conoce como espacio de estados. En una red de n nodos, el espacio de estados es de 2^n estados posibles. Esto significa que, conforme

aumenta el número de nodos de una red el espacio de estados crece exponencialmente y, por lo tanto, el número de simulaciones a realizarse de la red crece de igual manera.

Debido al crecimiento exponencial del espacio de estados en relación con el incremento en el número de nodos, la búsqueda de atractores en circuitos y redes biológicas ha resultado un problema de cómputo extremadamente demandante. De hecho, se ha demostrado en el ámbito de las ciencias computacionales que el tiempo de cómputo necesario para resolverlo aumenta exponencialmente conforme aumenta el número de nodos (Akutsu *et al*., 1998). Dentro de esta área, este tipo de problemas se clasifican como como problemas de tipo NP (*Nondeterministic Polynomial*) y se consideran como los más complejos dentro de las ciencias computacionales (Moret y Shapiro, 1991). Esto ha limitado de manera sustancial el cómputo de atractores en redes regulatorias. De hecho, mediante implementaciones basadas en software, el cálculo de todos los atractores cíclicos y puntuales correspondientes a la dinámica de una red se encuentra restringido a un número de nodos de alrededor de 20 (Berntenis y Ebeling, 2013; Irurzun-Arana *et al.*, 2016).

1.2.3 Enfoques de simulación por software

Existen dos grandes enfoques para calcular los atractores de una red. Por un lado, se puede explorar todo el espacio de estados en su totalidad, encontrando uno a uno los atractores dinámicos. Este enfoque basado en exploración exhaustiva se encuentra implementado en varias herramientas de software como GINsim (Chaouiya *et al.*, 2012), BoolNet (Müsel *et al*., 2010), y SPIDDOR (Irurzun-Arana *et al*., 2016), entre otras. Lamentablemente, la simulación exhaustiva está limitada a redes pequeñas (~20 nodos) dada la naturaleza exponencial de la demanda del tiempo de cómputo.

Por otro lado, existen métodos que tratan de superar esta limitante mediante técnicas heurísticas. Estas técnicas heurísticas son métodos que no garantizan una solución completa, pero si una solución parcial que puede resultar satisfactoria (Pearl, 1984).

En los párrafos siguientes, se describen algunas de las técnicas heurísticas más importantes para simular redes en la escala de decenas o cientos de nodos y se discutirán sus limitantes. En este trabajo, implementamos un método exhaustivo para encontrar todos los atractores de una red, sin utilizar técnicas heurísticas. En lugar de usar estas técnicas, aceleraremos el cómputo mediante un esquema basado en la implementación directa en hardware de los algoritmos de búsqueda de atractores, como se describirá en la metodología. Tal esquema nos permite romper la barrera de los 20 nodos en un tiempo considerablemente menor al de una búsqueda exhaustiva implementada en software, como se describe en la sección de resultados.

A la fecha existen varias herramientas de software para la identificación de atractores mediante la exploración exhaustiva del espacio de estados. En ellos se ha implementado una gama de algoritmos que permiten superar algunas de las limitaciones computacionales asociadas a la simulación de redes de mayor tamaño. Estos algoritmos simplifican la búsqueda de atractores usando alguna de las siguientes técnicas heurísticas: 1) Trasladar el sistema de ecuaciones Booleano a un formalismo matemático distinto en el que se simplifique su resolución; 2) Eliminar nodos en la red no relevantes para la dinámica; 3) Partir la red en redes de menor tamaño; 4) Acotar el espacio de estados a simular.

Heurística 1: Trasladar el sistema de ecuaciones Booleano a un formalismo matemático distinto

Para simplificar el problema de simulación, algunos algoritmos traducen el sistema de ecuaciones Booleanas que describen la dinámica de una red en diagramas de decisión binarios (He *et al.*, 2016; Ay *et al.*, 2009; Garg *et al.*, 2007). Estos son diagramas que representan a una función Booleana como un grafo, al cual se le pueden aplicar técnicas de manipulación simbólica para simplificar la función (Bryant, 1992). Al simplificar las funciones, se reduce el tiempo de cálculo de cada función. Sin embargo, estos algoritmos tienen la desventaja de generar posibles atractores espurios según el método de simplificación, requieren mucha memoria RAM, y no hay forma de determinar los requerimientos de memoria según el tamaño de la red (Dubrova y Teslenko, 2011). Este tipo de métodos se implementa en los programas genYsis (Garg *et al.*, 2008) y SQUAD (Di Cara *et al.*, 2007).

Otros algoritmos que utilizan el mismo principio se basan en métodos de satisfactibilidad Booleana (abreviados SAT). Estos convierten al sistema de ecuaciones Booleanas que describe la dinámica de la red en una sola función, G(x), de modo tal que el conjunto de valores para los cuales se satisface la condición G(x)=1 corresponden a estados dentro de atractores (Devloo *et al.*, 2003; Tamura y Akutsu, 2009; Dubrova y Teslenko, 2011). Este enfoque permite simular redes de hasta miles de nodos, pero requiere que las redes tengan ciertas características para poder ser usado. Por ejemplo, los algoritmos presentados tanto por Melkman *et al.*, (2010) como por Tamura y Akutsu (2009) solo son capaces de encontrar atractores en redes que estén compuestos de manera exclusiva por compuertas AND y OR, mientras que el algoritmo descrito en (Dubrova y Teslenko, 2011) está limitado a redes con interconectividad promedio menor a 2 conexiones por nodo, por lo que difícilmente este acercamiento puede tratar redes biológicas reales.

Heurística 2: Eliminar nodos

Existen algoritmos que eliminan de manera iterativa los nodos dentro de una red que no estén inmersos en un bucle de retroalimentación (Naldi *et al*.2011; Veliz-Cuba, 2011). Éstos permiten explorar la dinámica de redes de hasta más de 100 nodos, pero con la severa limitación de que sólo identifican atractores puntuales. Otra variante de este esquema de reducción consiste en la eliminación de nodos que tienen solo una entrada y una salida (llamados nodos mediadores) o que tienen el mismo valor en todos los atractores (Saadatpour *et al*., 2010; Saadatpour *et al*., 2011). Los algoritmos basados en este enfoque encuentran tanto atractores cíclicos como puntuales, pero en ciertos casos, por las restricciones de su uso, también tienden a generar atractores espurios (Berntenis y Ebeling, 2013).

Heurística 3: Dividir la red

Algunos algoritmos dividen las redes en subredes de menor tamaño e integran los resultados de los atractores locales de cada subred para obtener los atractores de la red completa (Zhao *et al*, 2013; Guo *et al.*, 2014). La partición de la red puede hacerse mediante criterios biológicos, por ejemplo, diferentes escalas de tiempo en la actividad de los componentes de la red (Quiñones-Valles *et al.*, 2014), o puede hacerse mediante el uso de algoritmos que permiten optimizar las particiones a partir de la topología misma de la red (Choo y Cho, 2015; Hong *et al.*, 2015); este enfoque en particular es implementado en el programa PAD (Hong *et al.*, 2015). Desafortunadamente en algunos casos las redes particionadas siguen siendo demasiado grandes para su simulación (Choo y Cho, 2015).

Heurística 4: Acotar el espacio de búsqueda

Finalmente, existen algoritmos cuya estrategia es evadir la simulación dinámica de la red partiendo de todos los estados iniciales posibles. Esto puede realizarse mediante métodos

estadísticos de muestreo (Zhang *et al.*, 2007), o mediante estrategias de teoría de grafos llamadas "Conjuntos mínimos de vértices de retroalimentación". Estas últimas, buscan el conjunto mínimo de estados iniciales a partir de los cuales se pueden obtener todos los atractores (Berntenis y Ebeling, 2013). La desventaja de estos algoritmos es que su eficiencia disminuye drásticamente conforme aumenta el grado de interconectividad de la red (Veliz-Cuba *et al.*, 2014).

Las herramientas de software actuales permiten al usuario realizar simulaciones exhaustivas de redes de regulación pequeñas con relativa simplicidad y tienen cierta flexibilidad respecto al uso de algoritmos heurísticos para abordar redes de mayor tamaño. Sin embargo, resulta relevante resaltar que la búsqueda exhaustiva de atractores en redes de regulación no reducidas de tamaño mayor a 20 nodos se encuentra sujeta al desarrollo de nuevos paradigmas algorítmicos y de plataformas computacionales de alto desempeño. En este trabajo, describimos nuestra contribución en este último aspecto. En particular, abordamos la búsqueda exhaustiva de atractores en redes Booleanas desde un paradigma de cómputo basado en una implementación algorítmica directa en hardware usando dispositivos conocidos como arreglos de compuertas programables en campo o *Field Programmable Gate Arrays* (FPGAs por sus siglas en inglés) Estos dispositivos están formados por una matriz de componentes electrónicos llamados elementos lógicos, cuya funcionalidad puede ser reprogramada para implementar distintas funciones Booleanas. Los elementos son programados al escribir un vector de bits de configuración en bloques de memoria contenidos en el FPGA.

Como se ha mencionado, utilizamos como caso de estudio dos redes de regulación reportadas en la literatura: que se describen a detalle en las secciones 1.3.1 y 1.3.2.

1.3.1 Red regulatoria asociada al control de ciclo celular en *Caulobacter crescentus*

C. crescentus es una bacteria Gram negativa que se divide asimétricamente dando origen a dos células hijas que desarrollan diferentes apéndices en cada uno de sus polos en etapas particulares del ciclo celular (Jensen, 2006; Wagner y Brun, 2007). Una de las células hijas, llamada *swarmer*, desarrolla un flagelo con el cual se mueve a través del medio líquido en búsqueda de nutrientes (Jensen, 2006). La otra célula hija, denominada *stalked*, presenta una extensión delgada de su cuerpo celular que se conoce como tallo, la cual le confiere la capacidad de adherirse a superficies sólidas. Una característica particular de esta división es que sólo la célula *stalked* es capaz de dividirse (Wagner y Brun, 2007), y la célula *swarmer* tiene que perder su flagelo y adquirir el fenotipo *stalked* antes de poder dividirse (England *et al.*, 2010). En cierto sentido, la división celular de *C. crescentus* se comporta de manera análoga a la división celular y la diferenciación en eucariotas. Por este motivo, se considera a esta bacteria como un organismo modelo para estudiar ambos fenómenos. La figura 4 ilustra el proceso de división celular asimétrica en *C. crescentus*.

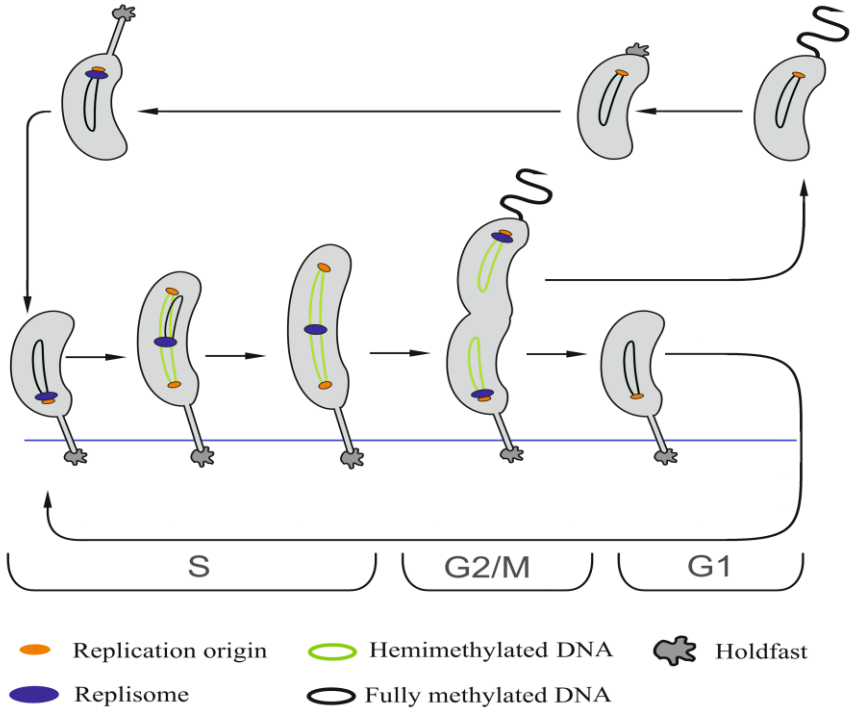

Figura 4. Ciclo de división celular en *C. crescentus*. La formación de las células *swarmer* y *stalked* se presentan en el contexto de la división celular. El ciclo celular es dividido en etapas análogas a la división celular en eucariotas. Figura tomada de Sánchez-Osorio *et al*, 2017.

La división celular asimétrica en *C. crescentus* es regida por una GRN conocida que está conformada por 153 genes (Fig. 5a), todos asociados al control de la replicación del DNA, la división celular, morfogénesis en los polos, quimiotaxis, etc. (Quiñones-Valles *et al.*, 2014). En el trabajo de Quiñones-Valles y colaboradores (2014) está GRN fue reducida mediante el algoritmo de Nadi *et al* (2009), resultando en una sub-red de 13 nodos, a la que se le denominó G_2 (Fig. 5b). Ésta sub-red fue posteriormente particionada en dos sub-redes de cinco y 8 nodos, llamadas respectivamente G_{2A} y G_{2B} (Fig. 5c y 5d). Esta partición se implementó porque G_2 contiene elementos cuyas interacciones operan a escalas de tiempo radicalmente diferentes. Específicamente, se incluyó en G_{2A} a aquellos elementos cuyas interacciones son mediadas por regulación transcripcional por TFs y por metilación del DNA, las cuales operan en escala de tiempo de minutos (Alon, 2006). Los otros nodos dentro de G_2 conforman una vía de señalización fosfo-proteolítica cuyas interacciones operan en el marco de milisegundos. Estos últimos nodos fueron asignados a la sub-red G_{2B}.

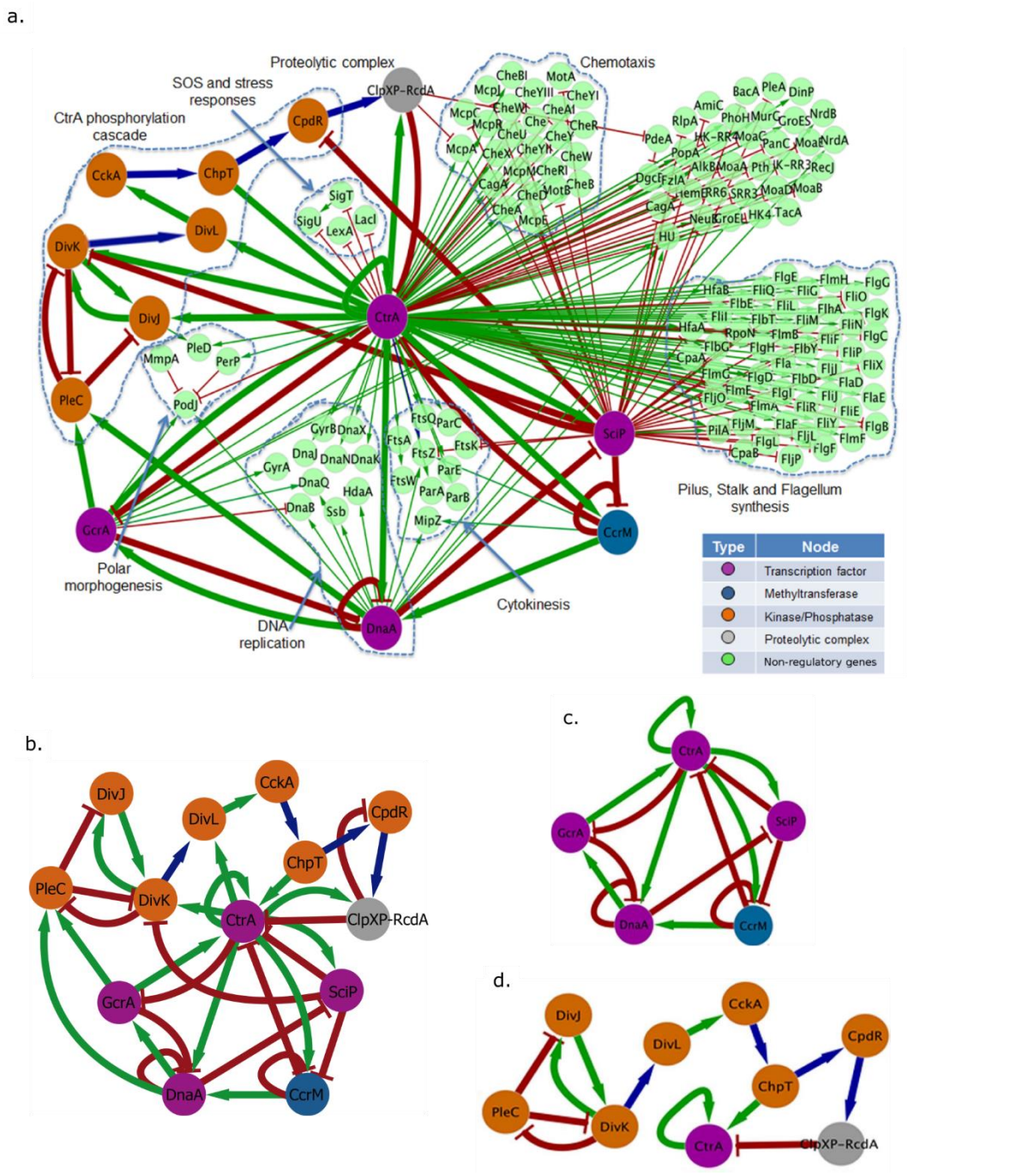

Figura 5. Red de regulación involucrada en el ciclo de división celular de *C. crescentus.* a) Red completa de153 nodos, G_1. b) Red reducida de 13 nodosG_2. c) Sub-red G_{2A}. d) Sub-red G_{2B}. Los colores de las flechas indican si se trata de una represión (roja), una activación (verde) o una interacción dual. Figura tomada de Sánchez-Osorio *et al*., 2017.

En este trabajo decidimos estudiar esta red debido a que describe la dinámica de la expresión de genes asociada al comportamiento de un organismo modelo, con reglas lógicas y atractores reportados en la literatura. Además, se trata de un modelo con el que nuestro grupo ha trabajado previamente (Sánchez-Osorio *et al*., 2017; Quiñones-Valles *et al*., 2014). Específicamente nos enfocamos en la sub-red G_{2A} dado que esta encapsula el control del ciclo celular de *C. crescentus* y es accesible a cualquier esquema de simulación dado su tamaño. A continuación, se describe el comportamiento de los reguladores que conforman esta red.

CtrA es un TF asociado a la regulación de alrededor de 100 genes involucrados en la formación del flagelo, el pili, y el aparato de quimiotaxis (Ryan y Shapiro, 2003). Además, este TF se une al ADN en el origen de replicación *oriC* del cromosoma de *C. crescentus* impidiendo el acoplamiento de la maquinaria de replicación del DNA y, en consecuencia, impide la replicación celular, principalmente durante la fase de *swarmer* (Quon *et al.*, 1998). CtrA presenta tanto autorregulación positiva como negativa, aunque la naturaleza de su autorregulación negativa no es bien entendida (Domian *et al.*, 1999). La transcripción de *ctrA* solo ocurre mientras que el cromosoma se encuentra hemimetilado, lo cual sucede en una ventana de tiempo después de la replicación del cromosoma. Una vez expresado, CtrA promueve la expresión de la metiltransferasa CcrM (Reisenauer y Shapiro, 2002). Esta enzima metila la segunda hebra del cromosoma de *C. crescentus* y, por lo tanto, reprime la expresión de CtrA (Reisenauer *et al.*, 1999a, 1999b). A nivel postraduccional, la actividad de CtrA es regulada por la proteína SciP, la cual secuestra este TF e impide que se una a sus blancos en el cromosoma (Gora *et al.*, 2010). La transcripción de *sciP* es promovida por CtrA e inhibida por DnaA (Tan *et al.*, 2010). DnaA es una proteína involucrada en la replicación del DNA, la cual se encuentra sujeta a varias formas de regulación; por un lado, su expresión requiere que el cromosoma este completamente metilado y, por lo tanto, su expresión es promovida por la actividad metiltransferasa de CcrM, por otro lado, su expresión es regulada positivamente por CtrA (Winzeler y Shapiro, 1996) y es inhibida por sí misma en un bucle de retroalimentación negativa (Collier *et al.*, 2006). DnaA promueve la expresión del TF GcrA (Winzeler y Shapiro, 1996), el cual adicionalmente es reprimido por CtrA (Holtzendorff *et al.*, 2004). Las reglas lógicas y los atractores de esta sub-red son descritos en las tablas 1 y 2 respectivamente.

Gen	**Regla lógica del gen en *t*+1**
CtrA	(CtrA OR GcrA) AND (NOT CcrM) AND (NOT SciP)
GcrA	DnaA AND NOT CtrA
SciP	CtrA AND NOT DnaA
DnaA	CtrA AND CcrM AND (NOT GcrA) AND (NOT DnaA)
CcrM	CtrA AND (NOT CcrM) AND (NOT SciP)

Tabla 1. Reglas lógicas correspondientes a los cinco reguladores de la sub-red $G_{2A.}$ Tomado de Quiñones-Valles *et al*., 2014.

	CtrA	**GcrA**	**DnaA**	**CcrM**	**SciP**	**Estado**
Atractor 1	0	0	1	0	1	I
	0	1	0	0	0	II
	1	0	0	0	0	III
	1	0	0	1	1	IV
Atractor 2	0	0	1	0	0	I
	0	1	0	0	0	II
	1	0	0	0	0	III
	1	0	0	1	1	IV

Tabla 2. Atractores reportados para G_{2A}. Se reportan dos atractores de ciclos de periodo cuatro. Tomado de Quiñones-Valles *et al*., 2014.

En la siguiente sección abordamos la otra red regulatoria estudiada en el presente trabajo.

1.3.2 Red de regulación genética involucrada en el proceso de formación de tumores en cáncer de colon asociado a colitis

Existe una correlación entre la inflamación intestinal no asociada a esteroides y la formación de tumores, particularmente en el caso de cáncer en el colon y recto (Garagnani *et al*., 2013; Balkwill y Mantovani, 2001; Giardiello *et al*., 1995; Ekbom *et al*., 1990). Se cree que esta correlación puede ser explicada a partir de la asociación entre las vías de señalización relacionadas a los procesos de inflamación y la proliferación celular. Esta idea surge porque se ha reportado que la desregulación de las vías de señalización asociadas a inflamación y al microambiente inmune contribuyen al desarrollo de cánceres (Lin *et al*., 2007; Yu *et al*., 2007; Greten *et al*., 2004; Bierie y Moses, 2010; Wang y Dubois, 2010). Además, se ha reportado que la inflamación vinculada a cáncer juega un papel importante en la resistencia a quimioterapia. Sin embargo, estas conexiones provienen de estudios que se enfocan solamente en una molécula o vía de señalización, y existe una carencia en la información acerca de cómo las vías de señalización involucradas en la inflamación interactúan con las vías clásicas asociadas a la formación de tumores.

Lu y colaboradores (2015) a partir de información experimental reconstruyeron un modelo Booleano de la red de regulación involucrada en la supervivencia y proliferación de células epiteliales pre-tumorales en un microambiente inflamatorio. La red resultante está conformada por 71 nodos (Fig. 6), pero debido a limitaciones computacionales fue reducida hasta 21 nodos (Fig. 7a) mediante el algoritmo de Saadatpour *et al* (2010). Su objetivo consistía en obtener un entendimiento holístico de los procesos involucrados en la formación de tumores asociados a inflamación. Esto permitiría el estudio sistemático de los mecanismos moleculares que subyacen al desarrollo del Cáncer de Colon Asociado a Colitis (CAC por sus siglas en inglés), además de la identificación de posibles blancos para considerarlos en el desarrollo de fármacos. Las reglas de la red se presentan en la tabla 3.

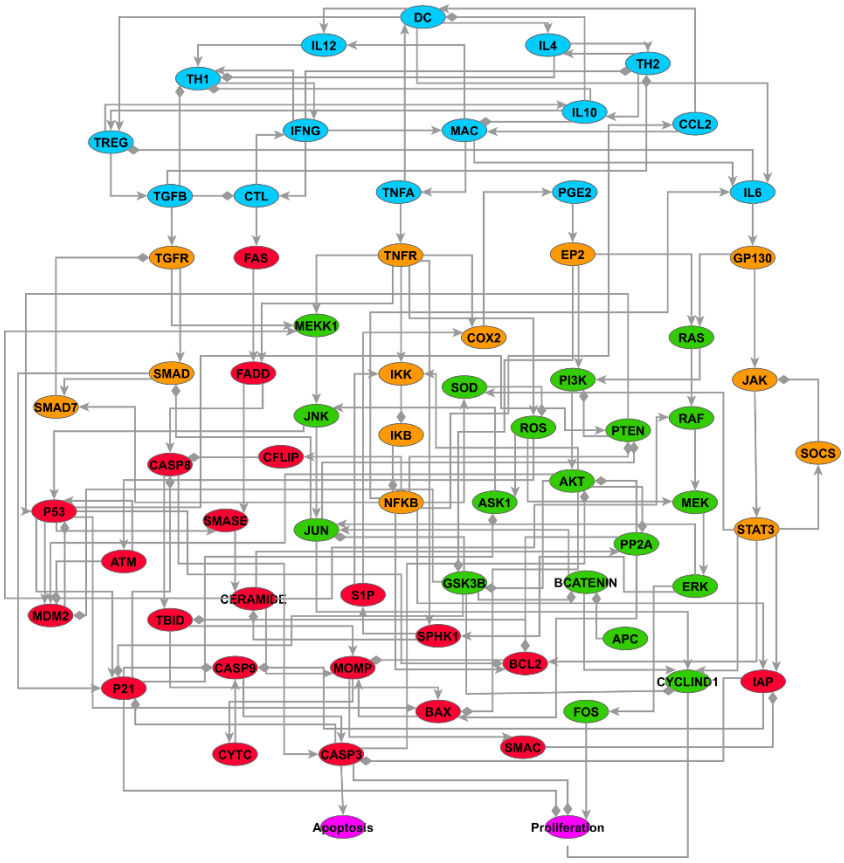

Figura 6. Red de regulación asociada a la supervivencia y proliferación de células pre-tumorales en CAC. Los nodos Azules representan señales del microambiente extracelular, los nodos naranjas son señales que participan principalmente en el proceso de inflamación, los nodos en verde son mediadores de la proliferación celular, y los nodos rojos están asociados a las vías de apoptosis. Los dos nodos morados representan el efecto fenotípico de las rutas dentro de esta red. Figura tomada de Lu *et al*., 2015.

Al analizar los atractores resultantes de la simulación de la red los autores identificaron un módulo regulatorio, formado por siete reguladores, que pudieran gobernar la transformación de células epiteliales en células pre-tumorales en el contexto de un microambiente inflamatorio (Fig. 7b). Más aún, realizaron estudios de perturbación *in silico* para identificar posibles blancos combinatorios, es decir, grupos de moléculas que deben de ser atacadas conjuntamente, para el desarrollo de cocteles de fármacos que inhiban la proliferación e induzcan la apoptosis en células tumorales. A partir de estas predicciones, realizaron experimentos *in vitro* para validar a las vías de señalización asociadas a ceramidas y a Pl_3K/AKT, obteniendo resultados de actividad antitumoral sinérgica significativamente mayores a la suma de sus efectos de los fármacos aplicados por separado. Se encontraron un total de 28 atractores, pero no se reportan sus dinámicas de manera directa. En lugar de ello se reportan las frecuencias del nodo de proliferación (Fig. 7c)

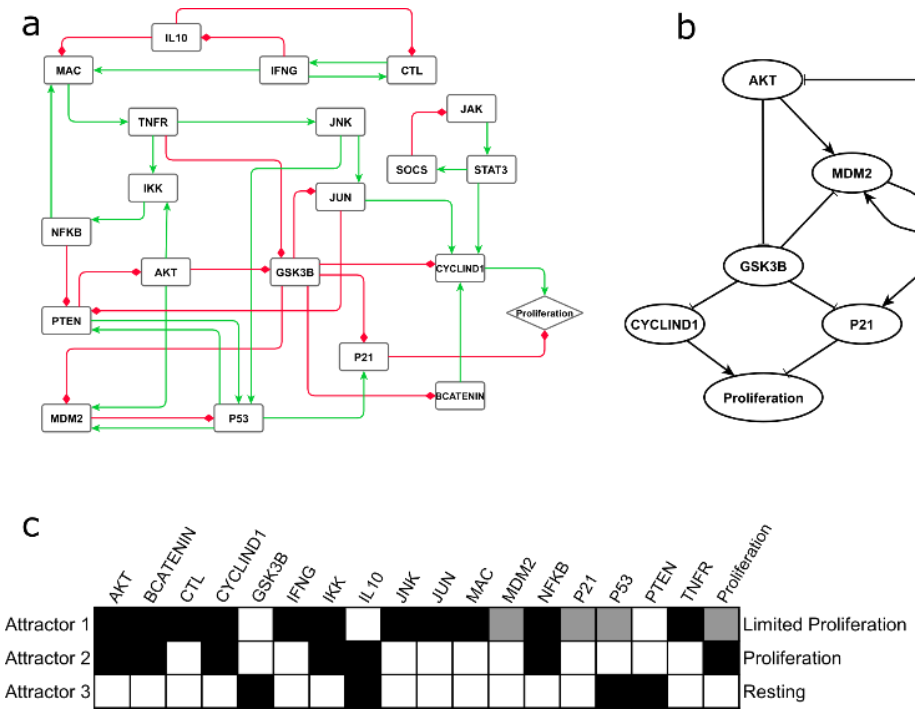

Figura 7. Red reducida y resultados del análisis de los atractores. a) Red regulatoria de CAC reducida hasta 21 nodos. b) Módulo regulatorio principal que determina el estado de la proliferación. c) Ejemplo de la representación por frecuencias para la red reducida. Figura tomada de Lu *et al*., 2015.

Cada fila en la figura 7C representa a un atractor en particular, y cada celda dentro de la fila representa a la frecuencia de activación de un gen en el atractor correspondiente. Las celdas de color negro significan que el nodo se mantiene encendido durante todo el atractor, las celdas de color blanco indican que permanece apagado en todos los estados, y las celdas de color gris apuntan a que el nodo se activa durante un porcentaje de la duración del atractor. Si el nodo de proliferación esta prendido todo el tiempo se clasifica en la categoría "*Proliferation*", si se mantiene apagado como "*Resting*", y si está prendido en un porcentaje se clasifica como de "*Limited proliferation*".

Gen	**Regla lógica del gen en *t*+1**
AKT	not PTEN
BCATENIN	not (P53 and GSK3B)
CTL	IFNG and not IL10
CYCLIND1	(BCATENIN or STAT3 or JUN) and not GSK3B
GSK3B	not (TNFR or AKT)
IFNG	CTL
IKK	AKT or TNFR
IL10	not IFNG
JAK	not SOCS
JNK	TNFR
JUN	JNK and not GSK3B
MAC	(IFNG or NFKB) and not IL10
MDM2	(P53 and AKT) and not GSK3B
NFKB	IKK
P21	P53 and not GSK3B

P53	(PTEN or JNK) and not MDM2
PTEN	P53 and not (NFKB or JUN)
SOCS	STAT3
STAT3	JAK
TNFR	MAC
Proliferation	CYCLIND1 and not P21

Tabla 3. Reglas lógicas de la red reducida para la formación de tumores en CAC (Lu *et al*., 2015).

II Objetivos

2.1 Objetivo general

Diseñar e implementar una plataforma de cómputo en hardware implementada en un dispositivo FPGA para simular la dinámica de redes regulatorias genéticas modeladas bajo un paradigma Booleano.

2.2 Objetivos específicos

1. Diseñar una arquitectura de hardware que emule la dinámica de un circuito genético. Este módulo debe de poder reproducir los resultados de la dinámica reportados en la literatura y permitir introducir perturbaciones a la dinámica de la red.
2. Diseñar e implementar una arquitectura genérica de hardware que encuentre mediante búsqueda exhaustiva a atractores en la dinámica de redes de regulación simulada por el módulo del objetivo 1.
3. Diseñar e implementar un sistema para almacenar el periodo de los atractores reportados por el módulo del objetivo 2.
4. Diseñar e implementar en sistema de comunicación, que permita el intercambio de datos entre el módulo del objetivo 3 y un microprocesador que se encuentra en el mismo chip que el FPGA.
5. Desarrollar una distribución de Linux que se ejecute en el microprocesador mencionado en el objetivo 4, la cual permita controlar al FPGA desde una interfaz en software.
6. Realizar evaluación comparativa del desempeño de una implementación en FPGA vs una implementación en software en la simulación de la dinámica de una red real de 21 nodos.

III Metodología

Se diseñó e implementó una arquitectura de hardware para la búsqueda de atractores en la dinámica de redes de regulación modeladas bajo un paradigma Booleano. En su forma más general, una arquitectura de hardware consiste en una descripción formal de los componentes físicos de un sistema y las interrelaciones entre dichos componentes. Esta descripción es organizada de manera tal que se permita entender la estructura y el comportamiento del sistema (Arora,, 2011). Se generaron dos variantes de la arquitectura de hardware, una diseñada para simular la dinámica de la red de regulación asociada al ciclo celular de *C. crescentus* (Quiñones-Valles *et al*., 2014), y otra para simular la dinámica de la red involucrada en la formación de tumores en CAC (Lu *et al*., 2015). La arquitectura elaborada consta de un conjunto de módulos que realizan cuatro funciones principales: i) ejecutar la simulación a nivel booleano, ii) buscar los atractores dinámicos, iii) guardar los resultados correspondientes, e iv) interactuar con un microprocesador para establecer comunicación con una interfaz de usuario en una computadora convencional. En cada una de las dos variantes de esta arquitectura, se desarrolló una distribución de Linux específicamente para configurar el FPGA de forma automática. Esta distribución se ejecuta en un microprocesador inmerso dentro del mismo chip en el que se encuentra el FPGA. Esto permite desplegar de manera más sencilla los resultados de las simulaciones a través de una consola.

3.1 Implementaciones en hardware reconfigurable: conceptos, principios y trabajo previo

En esta sección se introducen los fundamentos para el diseño de arquitecturas de hardware y su implementación en FPGAs, se detalla el proceso de implementación de un circuito que simula la dinámica de un circuito regulatorio simple, y se describen algunos trabajos previos realizados en este campo.

En el pasado, la implementación de diseños particulares en hardware requería del uso de componentes lógicos discretos en forma de circuitos integrados que implementaban funciones lógicas como AND, OR, NOT, etc. Estos tenían que ser conectados manualmente, asegurándose de mantener los niveles de voltaje apropiados para codificar estados de “prendido” o “apagado”. Esta práctica cambió radicalmente con la llegada de chips programables de alta densidad de componentes y el desarrollo de herramientas de Automatización de Diseño Electrónico (EDA por sus siglas en inglés). Estas herramientas son programas gráficos que se ejecutan en una computadora personal y que ofrecen a los desarrolladores los medios para diseñar e implementar circuitos con miles o hasta millones de compuertas lógicas.

Mediante herramientas de EDA comunes, un diseñador puede especificar e implementar arquitecturas de hardware en FPGAs usando Lenguajes de Descripción de Hardware (HDLs por sus siglas en inglés), que permiten describir una arquitectura de manera eficiente mediante un código que tiene características análogas a las de un lenguaje de programación. Sin embargo, esto no debe de ser confundido con un paradigma de software en el que se espera que las instrucciones dadas por el programador se ejecuten en un microprocesador, sino que se busca configurar elementos programables a nivel eléctrico dentro de un dispositivo de hardware reconfigurable para crear arquitecturas de cómputo de propósito específico.

3.1.1 Simulación de un solo nodo en una red biológica utilizando compuertas lógicas programables

Después de traducir el conocimiento biológico de una red a un modelo teórico (refiérase a la sección 1.1), el siguiente paso es resolver el modelo y derivar las implicaciones de sus resultados para un conjunto particular de condiciones. Sin embargo, como se argumentó en la sección 1.2.1, el problema del modelado de una red se vuelve inaccesible conforme aumenta el número de nodos. Respondiendo a la necesidad de extender el alcance de las simulaciones a redes grandes, se han propuesto nuevos paradigmas de cómputo basados en implementaciones en hardware para ejecutar de manera rápida la simulación de la dinámica de una red. Estas implementaciones se apoyan en el hecho de que las mismas compuertas lógicas empleadas para representar modelos Booleanos son coincidentemente los bloques funcionales más simples empleados en el diseño de arquitecturas de hardware. Percatarse de esto conlleva naturalmente a traducir un modelo Booleano de la dinámica de un gen a su realización física como un circuito electrónico concreto. Para simplificar las ideas que se desean exponer, el resto de esta sección se enfocará en la implementación de un solo gen, y en secciones subsecuentes se abordará una estrategia para extender este proceso a la escala de una red.

El primer paso es identificar una interacción biológica reportada, que haya sido bien caracterizada al punto de inferir su función Booleana correspondiente, y a partir de esto, avanzar hacia una descripción del circuito electrónico correspondiente que finalmente se plasma en un HDL. La figura 8 presenta dos casos hipotéticos de interacciones biológicas, y detalla la metodología para obtener una descripción apropiada para una implementación en hardware. En el primer caso, el gen A es reprimido por el gen B, de modo que su relación puede ser expresada por la compuerta lógica NOT, es decir el gen B se expresa en ausencia de A, y se silencia en su presencia. En el segundo caso tenemos que los genes A y B regulan al gen C, de forma tal que C solamente se expresa si A y B están presentes. Por esta razón, se puede definir a la interacción entre estos elementos mediante la compuerta AND.

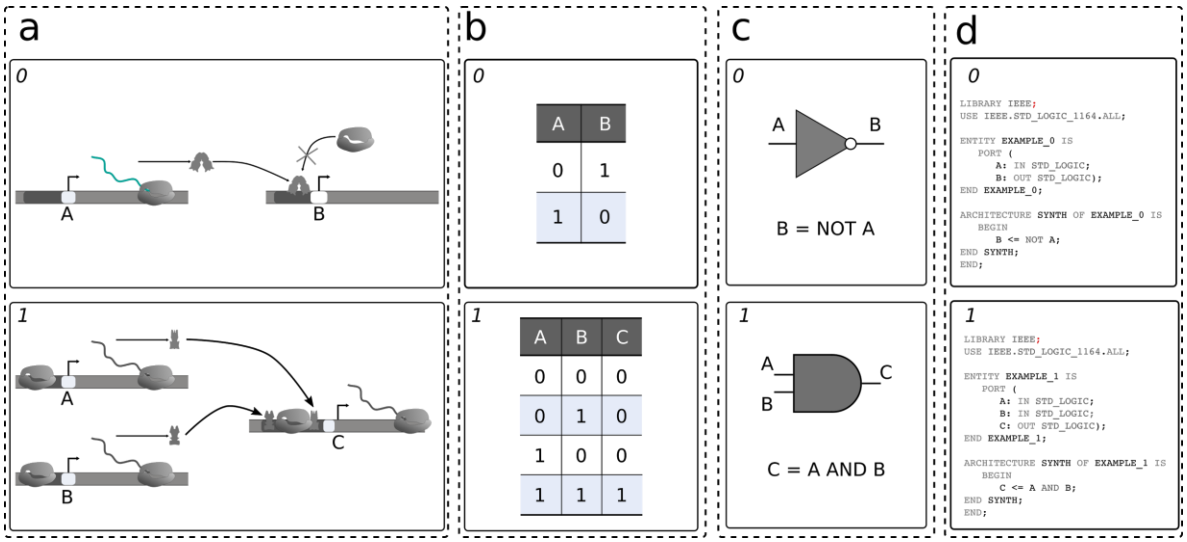

Figura 8. Flujo de trabajo para traducir una interacción regulatoria en una descripción de hardware. a) Interacciones biológicas conocidas. Se identifican interacciones biológicas a partir de literatura relevante o directamente mediante experimentos. b) Tablas de verdad. Se elabora una tabla de verdad para las interacciones. Estas tablas representan el comportamiento del gen de interés, visto como una variable de salida, en término de todos los genes que lo regulan (variables de entrada). c) Compuertas

lógicas. A partir de la tabla de verdad, se puede describir gráficamente el comportamiento del gen mediante el uso de compuertas lógicas. d) Descripción en HDL. Para cada función se elabora una descripción de hardware correspondiente. En este caso, se muestra la implementación de las interacciones hipotéticas en un HDL usado ampliamente en la academia conocido como VHDL (Very-High-Speed-Integrated-Circuit HDL).

Tras escribir el código de HDL correspondiente, se tienen que realizar una serie de procesos para generar una configuración que consiste en una serie de ceros (0) y unos (1) que permiten establecer las funciones e interconexiones de la arquitectura en un FPGA (Figura 9). La primera etapa de este proceso se llama síntesis lógica y consiste en convertir código HDL de alto nivel y descripciones de comportamiento del sistema en una simplificación equivalente en término de compuertas lógicas genéricas y la interconexión entre ellas. El resultado de este proceso es un archivo conocido como *netlist* (Vranesic Zvonko, 2013). El siguiente paso llamado mapeo de tecnología consiste en tomar esas compuertas y optimizarlas a los recursos tecnológicos disponibles del FPGA utilizado. Después del mapeo de tecnología sigue el proceso de colocación, el cual consiste en asignar los grupos de funciones generados en el mapeo de tecnología a elementos lógicos específicos en el FPGA. Tras la colocación sigue el proceso de ruteo, que consiste en configurar los bits de direccionamiento para interconectar los elementos lógicos de forma que sean congruentes con el diseño. Finalmente, en el proceso de generación de binario se obtiene un archivo que permite configurar el FPGA para implementar la arquitectura diseñada. Las plataformas de EDA actuales contienen herramientas para realizar todo este flujo de trabajo. Entre estas plataformas se encuentran Vivado® (Xilinx™), Synplify® (Synopsis™) y Quartus II® (Intel™). Esta última fue la que se utilizó en el presente trabajo.

Una vez que el diseño ha sido sintetizado, el FPGA puede adoptar la arquitectura deseada cargando el archivo binario correspondiente. Por lo general, este archivo puede ser cargado desde un microprocesador, o puede ser leído desde una memoria flash o ROM. En cualquier caso, un archivo de configuración apropiado debe ser cargado al FPGA cada vez que este arranque, así como cada vez que el usuario desee cambiar la arquitectura. Una vez configurado, un FPGA opera como un dispositivo digital dedicado.

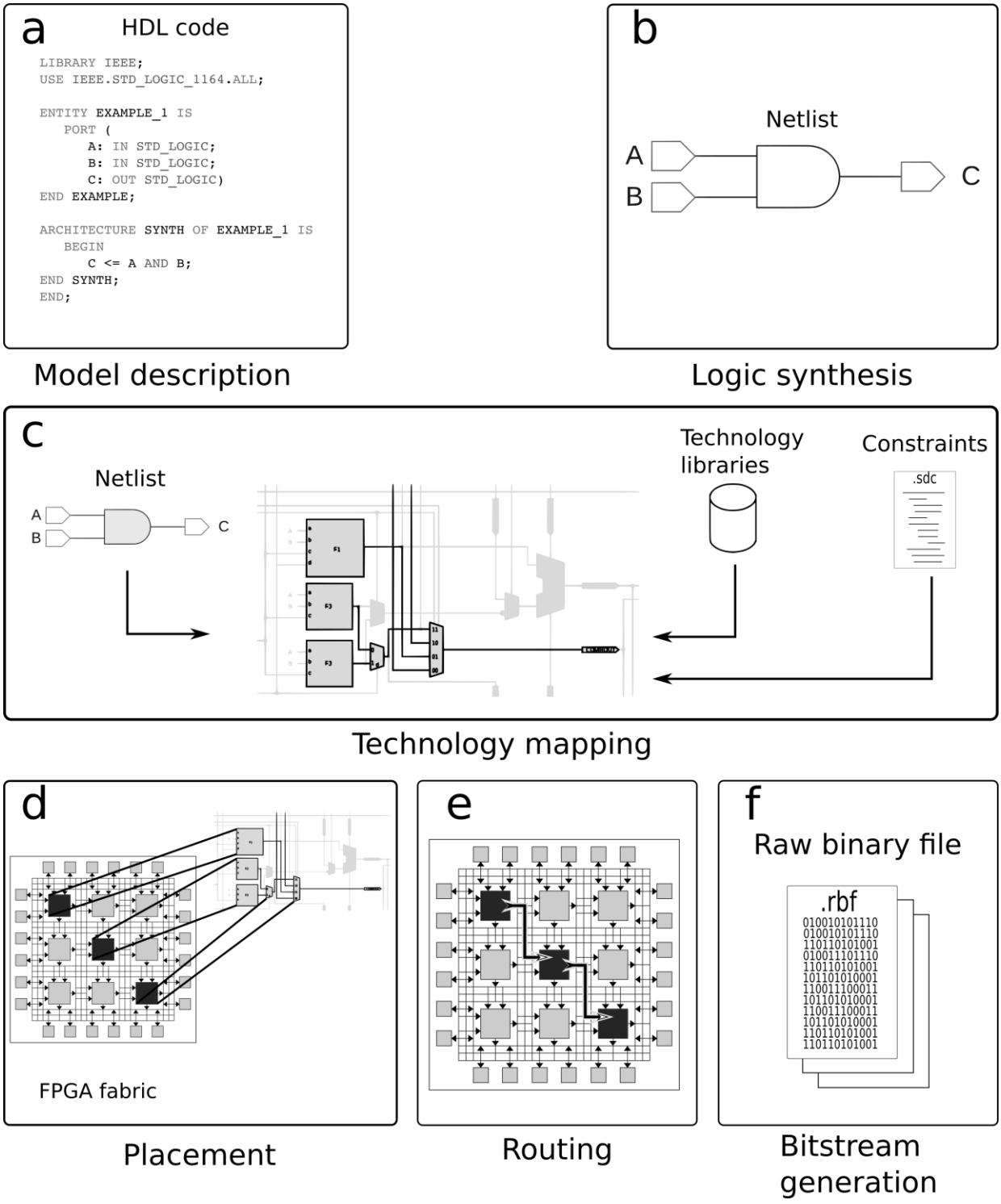

Figura 9. Esquema del flujo de trabajo para configurar un FPGA para una arquitectura específica. a) Descripción del modelo. Se parte de una descripción en un HDL de la arquitectura que se desea simular. b) Síntesis lógica. Se genera una representación genérica de la arquitectura en términos de compuertas lógicas. c) Mapeo de tecnología. Las funciones lógicas se agrupan según los recursos disponibles en el FPGA. Por un lado, mediante las bibliotecas de tecnología se ajustan al tipo y número de los elementos lógicos disponibles en el circuito. Por otro lado, se ajustan para cumplir con restricciones específicas definidas por el usuario. d) Colocación. Los grupos de funciones ajustados se asignan a elementos lógicos específicos en el FPGA. e) Ruteo. Se configura la conexión entre elementos lógicos. f) Generación de binario. Se obtiene el archivo de configuración para implementar la arquitectura en el FPGA.

3.1.2 De un solo gen a un circuito: llevando la simulación en hardware al siguiente nivel

Siguiendo la metodología descrita en la sección pasada, es posible extender una implementación de hardware para representar a un conjunto pequeño de genes con interacciones regulatorias. Para una red, el estado del sistema completo en un momento particular se puede expresar como un vector de elementos binarios de un bit, en el cual el *i-ésimo* bit corresponde al estado del gen *i*. El proceso de simular la dinámica de una red consiste en empezar desde un estado inicial, y luego actualizar el sistema de manera iterativa. Conforme pasan las iteraciones, el estado del sistema transita por una trayectoria, que eventualmente converge en un atractor que puede ser cíclico o puntual.

Para implementar una red en hardware, primero hay que tener las descripciones apropiadas de cada uno de sus nodos, como se estableció en la sección anterior. El siguiente paso es conectar los nodos de forma coherente con la topología del circuito simulado. Esta etapa requiere especificar la interconexión de los nodos dentro de la descripción de la arquitectura de hardware. Los HDLs actuales permiten diseñar componentes que integran dentro de sí a otros componentes. Para hacer esto, el código de una arquitectura se importa desde otro archivo la descripción del componente que se desea incorporar, de manera análoga a como el código de un programa de software importa bibliotecas. Para cada componente instanciado, se debe especificar el número, tipo, dirección y ancho de sus puertos, de manera que se puedan conectar con la arquitectura principal fácilmente y de manera reproducible. Un mismo componente puede ser llamado varias veces dentro de una arquitectura, conectando cada copia con diferentes partes del diseño.

La figura 10, muestra un ejemplo de un diseño modular que emplea el mismo módulo varias veces en la misma arquitectura. En este ejemplo hipotético, el dispositivo “example” posee cuatro entradas de cuatro bits llamadas “d0”, “d1”, “d2”, y “d3”, así como una sola salida de cuatro bits llamada “Y”. El comportamiento de este dispositivo puede considerarse como una función de dos entradas de cuatro bits, y una salida del mismo ancho, la cual es implementada tres veces. Primero para el primer par de entradas, luego para el segundo par de entradas, y luego para las salidas de las dos primeras implementaciones. La función es implementada por un dispositivo llamado “mod”, que es llamado cada vez que se requiera implementar la función, adoptando un nombre específico para cada instancia.

a

```
module example(input logic [3:0] d0, d1, d2, d3,
               output logic [3:0] y);

  logic [3:0] low, high;
  mod down(d0, d1, low);
  mod up(d2, d3, high);
  mod final(low, high, y);
endmodule
```

b

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity example is
  port(d0, d1,
       d2, d3: in STD_LOGIC_VECTOR(3 downto 0);
       y: out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture struct of example is
  component mod
    port(d0,
         d1: in STD_LOGIC_VECTOR(3 downto 0);
         y: out STD_LOGIC_VECTOR(3 downto 0));
  end component;

  signal low, high: STD_LOGIC_VECTOR(3 downto 0);

begin
  down: mod port map(d0, d1, low);
  up: mod port map(d2, d3, high);
  final: mod port map(low, high, y);
end;
```

c

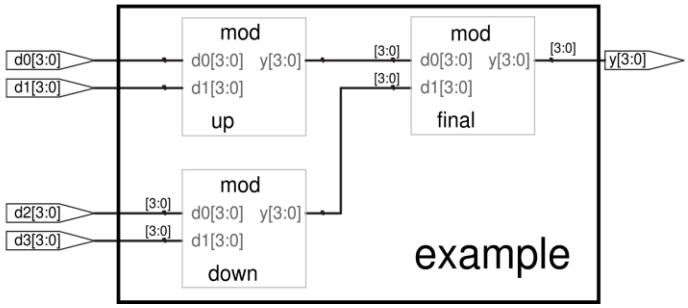

Figura 10. Instanciación de módulos en un diseño de hardware. Los recuadros a y b presentan el código fuente para el dispositivo "example" en los lenguajes Verilog y VHDL respectivamente. En ambos casos, el nombre del dispositivo principal, así como las características de sus entradas y salidas, son declaradas en un solo enunciado, el cual se llama "module" en Verilog y "Entity" en VHDL. Las conexiones internas dentro del diseño son declaradas en Verilog con el enunciado "logic" y el enunciado "Signal" en VHDL. Nótese que para declarar los dispositivos instanciados y especificar sus conexiones, VHDL ocupa dos enunciados llamados respectivamente "component" y "port map". En contraste Verilog hace ambas cosas en un solo enunciado de instancia. c) Representación del circuito "example", que muestra el circuito resultante de compilar cualquiera de los dos códigos.

Desde el contexto de la implementación en físico, lo que hacen las plataformas de EDA es asignar las funciones de cada componente instanciado a uno o más elementos lógicos durante el mapeo de tecnología y la colocación, y luego en el proceso de ruteo configura los elementos de direccionamiento para implementar las conexiones descritas en puertos y señales en el HDL.

En el resto de esta sección, a modo de ejemplo y para consolidar la metodología descrita anteriormente, se presentará una estrategia de implementación de un circuito biológico tomando como ejemplo el circuito sintético conocido como "*repressilator*" (Elowitz y Leibler, 2000). Como se ilustra en la figura 11, este circuito está conformado por un bucle de regulación formado por tres genes que codifican para TFs que reprimen al siguiente nodo en el bucle.

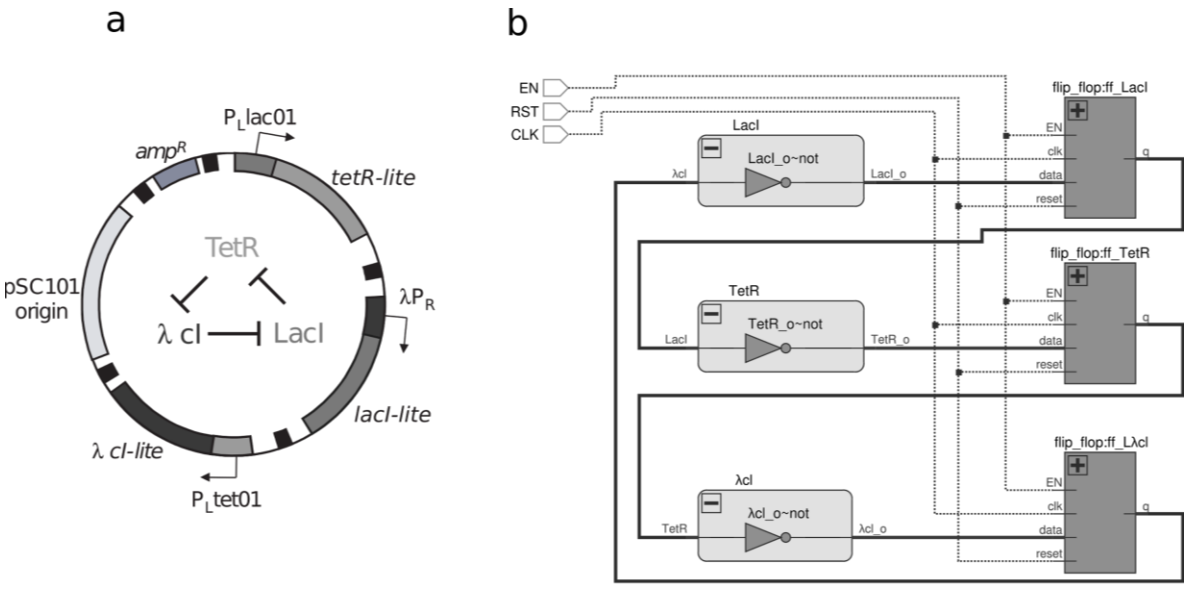

Figura 11. Implementación de un circuito genético simple como una arquitectura de hardware. a) Representación del modelo de la red y mapa del plásmido para el *repressilator*. El regulador LacI se une al promotor PLlac01, inhibiendo la expresión de *tetR*. TetR a su vez se une al promotor PLtet01, lo que reprime la expresión del gen *λcI*. El TF λCI se une al promotor λPR, lo que reprime a LacI y cierra el bucle de regulación. Modificado del trabajo de Elowitz y Leibler (2000). b) Circuito electrónico que implementa el comportamiento del *repressilator*. El circuito consta de tres nodos que implementan las funciones Booleanas de los genes. A cada uno de estos nodos de procesamiento corresponde un elemento llamado *flip-flop*, que almacena el estado actual del nodo. Las líneas punteadas representan a señales de entrada que se requieren para controlar los *flip-flop*.

Como se muestra en la figura 11, además de implementar la interconexión de los nodos que implementan las funciones, es necesario incorporar elementos que permitan almacenar el estado del nodo en un tiempo determinado. En el campo de la electrónica digital, tales elementos se implementan mediante dispositivos llamados *flip-flops* De manera resumida, estos dispositivos guardan el valor de un bit, actualizando el mismo solamente en el flanco de subida de una señal oscilatoria con onda cuadrada., a la cual se le llama reloj. Un componente de este tipo debe de ser agregado a la salida de cada elemento de procesamiento para poder guardar su estado. Por lo general, se utiliza la misma señal oscilatoria para todos los elementos de procesamiento del diseño, de modo que un solo reloj maneja la frecuencia con la que se actualiza el sistema completo. La figura 12 detalla el funcionamiento de estos dispositivos.

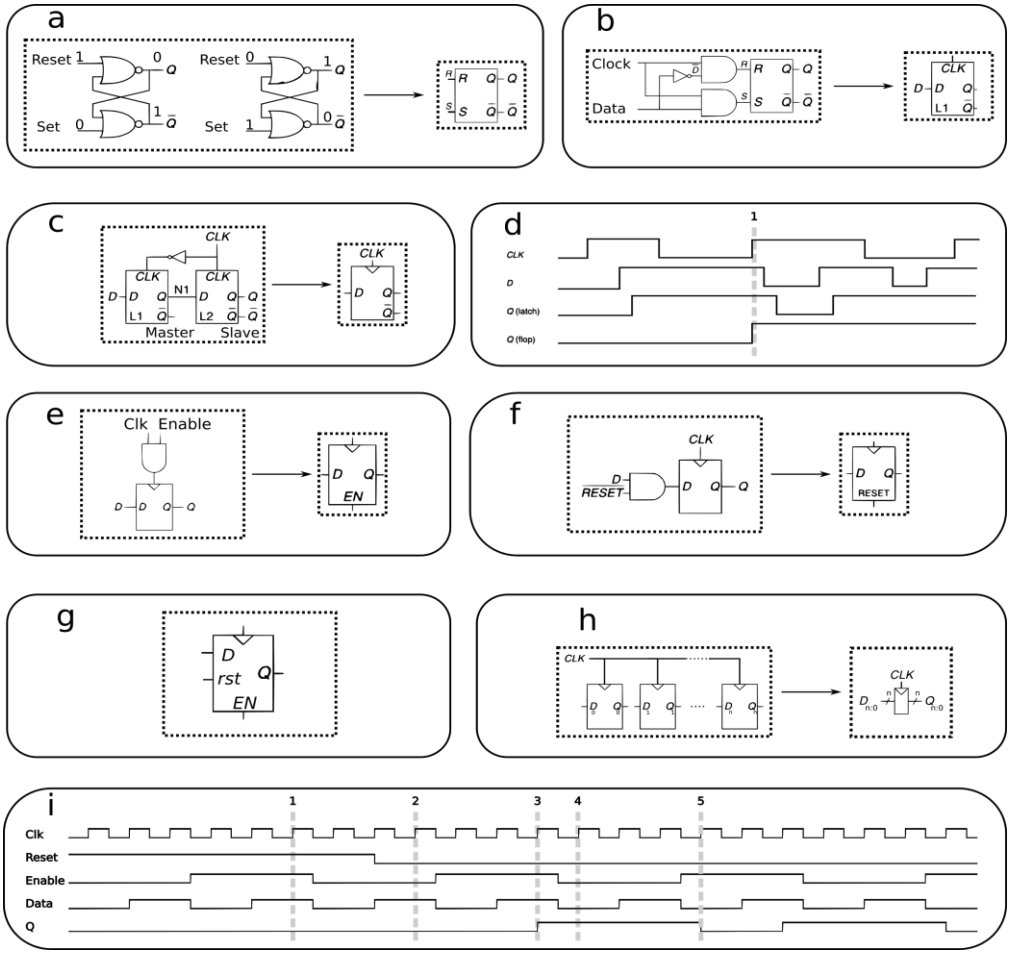

Figura 12. Diagrama del funcionamiento de un *flip-flop* desde 0. a) SR-Latch. Un *latch* es un dispositivo que guarda un solo bit de información mediante un bucle que permite mantener invariante el valor de su salida. Un SR-Latch, es un tipo de latch específico que tiene una entrada de "set" que cambia su salida a 1, una entrada "*reset*" que lo cambia a 0. b) D-Latch. Un D-Latch es un SR-latch con un módulo extra a la entrada que permite usar una sola entrada de datos, D, en lugar de dos entradas Set y *Reset*. Adicionalmente agrega una entrada de reloj que le indica cuándo actualizar el sistema, permitiendo que este cambie mientras que la entrada de reloj valga 1. c) D-*Flip-flop*. Al conectar dos D-Latches en serie, e invertir el valor de la señal de reloj en el primer Latch, Este diseño garantiza que el valor guardado solo se actualice en el flanco de subida, en lugar de todo el tiempo que el reloj esté encendido. Esto permite minimizar el ruido causado por cambios en el sistema mientras el reloj está en uno. d) Comparación de la dinámica de un D-*Flip-flop* y un D-latch. Se remarca con una línea gris punteada el momento en el que el *flip-flop* actualiza su estado. e) *Flip-flop* con *enable*. Se expande el modelo del D-*Flip-flop* para permitir que un *flip-flop* solo se actualice en ciertos casos flancos de subida del reloj, que se indican con la señal "*enable*". f) *Flip-flop* con *reset*. Se expande el D-*Flip-flop* para permitir al diseñador tener una forma de enviar al *flip-flop* a "0" sin importar las condiciones de las demás entradas. g) *Flip-flop* con *enable* y *reset*. Se combinan los dos diseños anteriormente descritos para tener un control fino del funcionamiento del *flip-flop*. h) Registro. Este dispositivo consiste en una serie de *flip-flops* controlados por un mismo reloj, y se utiliza para almacenar valores de más de un bit. i) Onda de un *flip-flop* con *enable* y *reset*. Cada una de las líneas punteadas remarca un caso particular del comportamiento de este dispositivo en un flanco de subida de reloj. En la línea 1, la salida q se mantiene en "0" porque la señal de *reset* está activa. En la línea 2, el valor se mantiene invariante dado que, aunque *reset* está apagado, la señal de *enable* no está prendida. En la línea 3, el valor de q cambia a 1, dado que *reset* es "0", *enable* es "1" y data es "1". Finalmente, en la línea 5 q regresa a "0" porque *enable* es "1" y data es "0".

En la mayoría de los FPGAs actuales, los elementos lógicos incorporan *flip-flops* organizados en registros como componentes dedicados. Las herramientas de síntesis en las plataformas de EDA detectan cuando un *flip-flop* es declarado en una descripción de hardware, y los asignan de manera automática a dichos registros. Esto permite un diseño más compacto que utiliza los elementos reconfigurables exclusivamente para implementar funciones del usuario.

Además de sincronizar el sistema, algunos *flip-flops* incorporan señales de *reset* y habilitación (Fig. 12) que permiten al diseñador asegurarse de que los estados solo serán cargados en momentos específicos en lugar de en cualquier ciclo de reloj, y llevar los valores a "0" en cualquier momento. En términos de una simulación de red, una señal de habilitación puede ser usada para asegurarse de que el sistema empiece a ejecutar la simulación de la red hasta después de haber cargado los estados iniciales, mientras que el *reset* puede servir para limpiar los valores guardados antes de empezar una nueva simulación, o simplemente para llevar al sistema a un estado bien definido.

Para poder entregar estas señales en los momentos apropiados, se debe de incorporar un tipo de módulo que coordine la funcionalidad del sistema con la menor intervención del usuario posible. El tipo de elemento que satisface esta necesidad es conocido como Máquina de Estados Finitos (FSM por sus siglas en inglés). Estos elementos son circuitos síncronos secuenciales, lo que quiere decir que son circuitos con registros controlados por un solo reloj, cuyas salidas dependen no solo del valor actual de sus entradas, sino además de su propio comportamiento en el pasado. El nombre máquina de estados finitos, surge del hecho de que su comportamiento funcional puede ser representado como un número finito de estados, como se presenta en la figura 13.

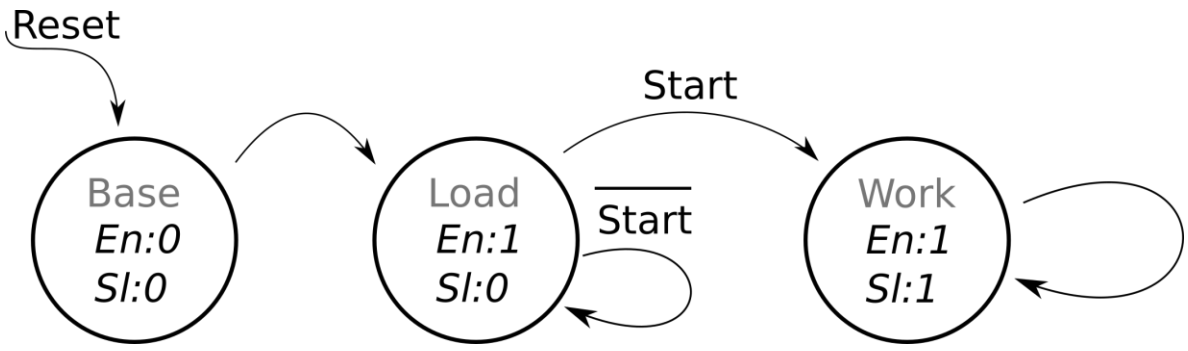

Figura 13. Diagrama de estados de una máquina de estados finitos. Se presenta un ejemplo de FSM con tres estados llamados *Base*, *Load* y *Work*. La máquina tiene además una salida llamada *Start* y dos salidas llamadas En y Sl. Cada uno de los círculos en el diagrama muestra el nombre del estado y el valor de sus salidas en ese estado en particular. Cada uno de los arcos representa la transición entre estados dada una condición, la cual es especificada por las señales que se encuentran junto al arco. En caso de que un arco regrese a su mismo origen, significa que esa condición mantiene al sistema en el mismo estado. Las condiciones en los arcos son expresadas por el nombre de las señales que deben estar prendidas o apagadas según sea el caso, si el nombre de la señal tiene una raya encima significa que la condición es que la señal sea "0", de lo contrario, significa que la señal debe de valer "1". Cuando un arco no tiene condición, significa que la transición se hace de manera automática en el siguiente ciclo

de reloj sin importar los estados de las entradas. Este diagrama en particular corresponde a un tipo de FSM conocido como máquina de Moore.

Un concepto que surge al considerar a las máquinas de estado como módulos que coordinan al sistema, es que podemos dividir nuestros diseños en dos grupos de componentes separados. Por un lado, tenemos a todos los módulos que se encargan de procesar y almacenar los datos. Por otro lado, tenemos a los circuitos secuenciales que se encargan de recibir instrucciones y coordinar al sistema. A estos dos grupos se les llama respectivamente "*datapath*" y "unidad de control". En conjunto, estos grupos permiten diseñar circuitos que puedan procesar datos de manera automática dado un conjunto de instrucciones simples. La figura 14, muestra una implementación en hardware de la dinámica del *repressilator*, que de manera automática carga un estado inicial y realiza la simulación.

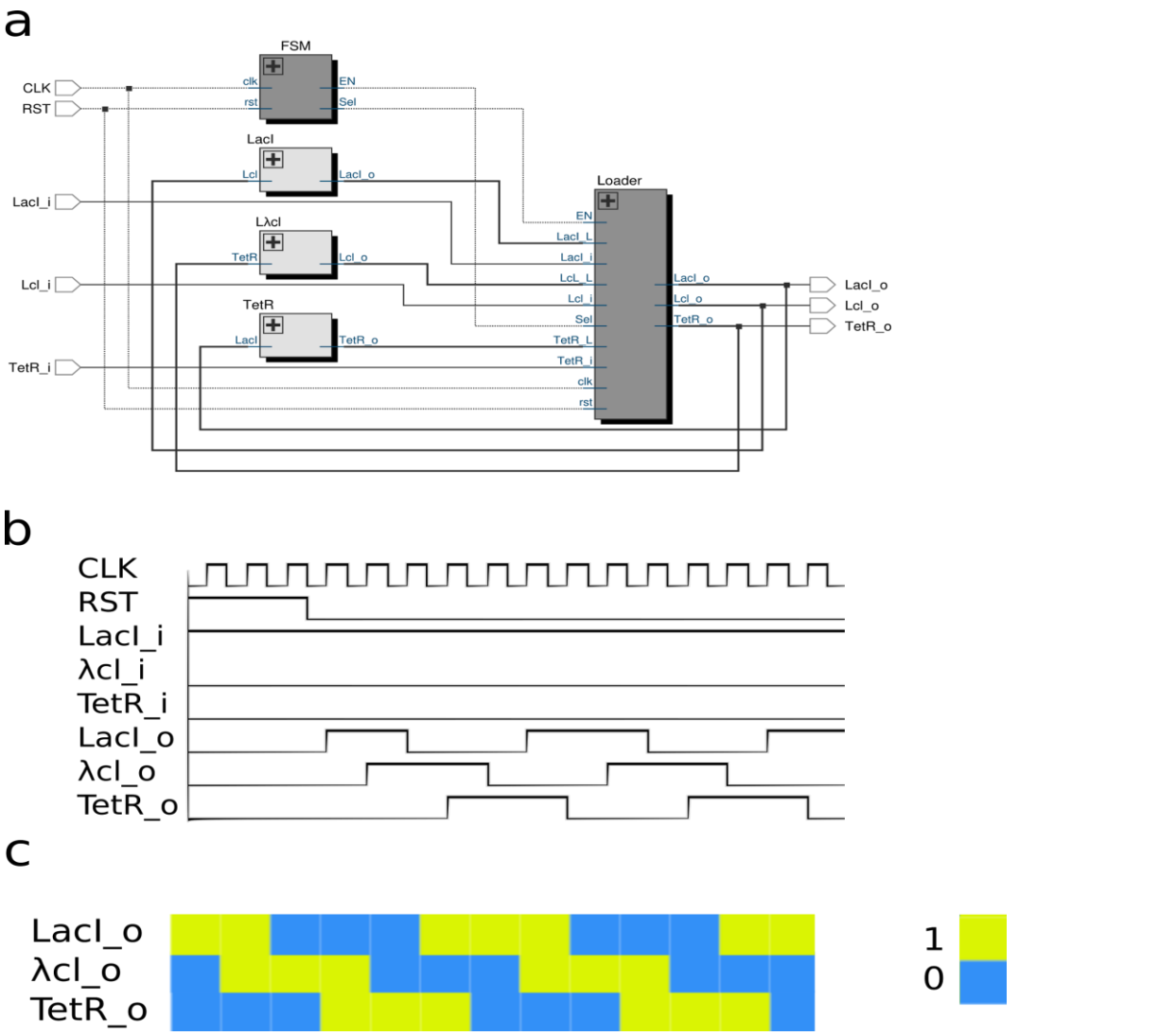

Figura 14. Implementación en hardware del *repressilator*, incorporando una unidad de control. a) Diagrama de la arquitectura de hardware. Como se hizo en el diseño presentado en la figura 11, se implementan tres unidades de procesamiento que implementan la función de cada uno de los genes en la red (rectángulos color gris claro), cuyas interconexiones (líneas gruesas) pasan por un módulo llamado *Loader* (gris oscuro), formado por a tres *flip-flops* con *reset* y *enable*. Este módulo recibe tres

entradas llamadas Laci_i, λcl_i, y Petri, que corresponden a los estados iniciales de cada nodo, y además recibe a tres señales llamadas Laci_o, λcl_o, y TetR_o, que corresponden a las salidas de los módulos de procesamiento. Adicionalmente, *Loader* recibe una señal de *enable*, una señal de *reset*, y una señal llamadas "sel" que le indica si cargar el estado inicial, o el estado que venga de los módulos de procesamiento. Tanto sel como *enable* provienen de una máquina de estados en lugar del exterior. b) Resultados de la simulación de la implementación en hardware del *repressilator*, a partir de los estados iniciales LacI_i = 1 λcl_i = 1 y TetR_i = 1. Mediante la coordinación de la FSM, el sistema logra coordinar la carga del estado inicial y el inicio de la simulación solamente con el *reset* y el pulso de reloj. c) Simulación del *repressilator* mediante el software BoolSim (Bock et al, 2004), partiendo del mismo estado inicial que el hardware. Ambas implementaciones arrojan el mismo comportamiento.

3.1.3 Trabajo previo

En el trabajo de Miskov-Zivanov y colaboradores (2011a), se implementó un circuito de hardware que emula a la red de señalización asociada a la diferenciación en células T, reproduciendo la dinámica reportada para una implementación en software, solo que 60,000 veces más rápido. En un trabajo posterior, este mismo grupo extendió esta estrategia a la simulación de más redes biológicas (Miskov-Zivanov *et al.*, 2011b). Sin embargo, sus implementaciones no incorporan una unidad de control, y por lo tanto son incapaces de realizar la búsqueda automática de atractores. Sus implementaciones requieren ser configuradas para ejecutar la simulación por un número determinado de iteraciones a partir de un estado inicial. Además, sus métodos de entrada y salida de datos son poco prácticos, pues los resultados se despliegan en un arreglo de LEDs y utilizan interruptores para su configuración. Posteriormente propusieron una metodología para el diseño automatizado de modelos Booleanos a partir de datos biológicos (Miskov-Zivanov *et al.*, 2013) y sugirieron la posibilidad de automatizar el proceso de implementación en FPGAs, sin embargo, no presentan un ejemplo concreto de esta aplicación.

Un diseño como el presentado en la figura 14 puede ser extendido a la implementación de redes de cualquier tamaño, con la única limitante de espacio físico que consuma el diseño en el FPGA. Por ejemplo, Zerarka y colaboradores (2004) implementaron un diseño similar para una serie de redes Booleanas generadas aleatoriamente con tamaños de hasta 800 nodos. Los autores trabajaron con redes aleatorias, dada la carencia de redes biológicas a esas escalas con reglas reportadas. Realizaron una evaluación comparativa contra una implementación en Matlab corriendo sobre un CPU, comparando el tiempo que les tomaba correr una dinámica por 100,000 iteraciones. Su implementación mediante FPGA logró superar la velocidad de la implementación del software por hasta tres órdenes de magnitud. Adicionalmente, incorporaron un módulo de búsqueda de atractores, que permite recuperar un atractor a partir de un estado inicial, pero no tiene forma de realizar una exploración automática del espacio de estados. Pese a sus limitaciones, este trabajo demostró que es posible simular la dinámica de una red en hardware programable, y que este esquema de simulación es más eficiente que los esquemas basados en software.

Desafortunadamente, los beneficios de las implementaciones en hardware se ven opacados por el tiempo que toma reconfigurar un dispositivo para un diseño en particular. Más aún, la implementación presentada por Zerarka et al (2004) es totalmente fija, lo que significa que no presenta manera alguna de cambiar ni las conexiones ni reglas de la red. Eso implica que una

nueva arquitectura tiene que ser diseñada e implementada cada vez que uno quiera trabajar en una nueva red, o evaluar las implicaciones de un cambio en la red sobre la dinámica de su expresión.

Esto pone a las implementaciones de hardware en desventaja contra las implementaciones basadas en software, pues, aunque la ejecución de un algoritmo pueda hacerse en la escala de tiempo de milisegundos, tener que volver a diseñar una arquitectura de hardware puede tardar semanas, y el proceso de depurar el diseño puede tardar varias horas cada vez que se tenga que sintetizar la arquitectura (Hung y Wilton, 2015).

Una solución para reducir la demora generada por el proceso de síntesis consiste en el uso de superposición en el FPGA. Esta estrategia consiste en implementar una arquitectura reconfigurable encima de la capa física del FPGA, de modo que el diseño solo requiera ser sintetizado una vez (So y Liu, 2016). Esto permite realizar cambios en el funcionamiento de la arquitectura implementada sin necesidad de repetir el proceso de síntesis, permitiendo hacer cambios en segundos a costa de aumentar el espacio que el diseño ocupa en el FPGA.

Hasta donde nuestro conocimiento abarca, el único trabajo publicado que ha empleado superposición para la búsqueda de atractores en FPGAs a la fecha es el de Ferreira y Vendramini (2008), quienes implementaron sobre un FPGA una arquitectura que permite agregar o eliminar conexiones entre nodos. Para esto generaron de manera aleatoria un conjunto de componentes que realizan la función de nodos con diferentes grados de interconexión. Estos componentes se enlazan entre sí mediante un módulo reconfigurable conocido como Red de Interconexión Multi-etapa. Estos módulos consisten en un arreglo regular de conmutadores que permiten cambiar la dirección o interrumpir el paso de una señal a través del circuito (Kraetzl y Hsu, 1999). Este diseño permite abordar la búsqueda de atractores en redes de distintas topologías sin tener que recompilar un diseño específico para cada topología. Cabe remarcar que los trabajos mencionados anteriormente abordaron redes con topologías aleatorias, es decir redes en las cuales todos los nodos tienen el mismo número de interconexiones (Kauffman, 1969). Al incluir nodos con diferentes grados de interconectividad, el diseño de Ferreira y Vendramini permite abordar redes con topologías libres de escala, es decir redes en las que la mayoría de los nodos tienen muy pocos vecinos dentro de la red, pero hay un grupo reducido de nodos altamente interconectados (Barabasi y Oltavi, 2004). La relevancia de esta característica radica en que, en la naturaleza, la mayoría de las redes presentan topologías libres de escala (Watts y Strogatz, 1998; Barabasi y Oltavi, 2004).

Otro aspecto importante que considerar en las implementaciones en FPGAs es cómo conectarlos con una interfaz que nos permita enviar y recibir datos, además de enviar las configuraciones necesarias para una simulación correcta, especialmente en los diseños con superposición. La manera más simple, desde el punto de vista del diseño de hardware, es simplemente conectar el FPGA con una interfaz física (botones, interruptores, LEDs etc.), como se hizo en los trabajos de Miskov-Zivanov y colaboradores (2011a; 2011b). Sin embargo, este enfoque resulta poco práctico en redes de más de unos cuantos nodos, y no ofrece forma alguna de recuperar los resultados. Ferreira y Vendramini (2008) realizan esta función con un procesador "softcore", es decir un microprocesador de propósito general implementado en el FPGA, sobre el cual se ejecuta código en "*bare metal*" enviado desde la computadora mediante USB o algún protocolo de transferencia serial. Por otro lado, Zerarka y colaboradores (2004)

emplearon una plataforma de cómputo tipo Sistema en un Chip (SoC por sus siglas en inglés). Esta plataforma consiste en un FPGA y un microprocesador físicamente dentro de un mismo chip, y conectados directamente para minimizar el retraso en la transferencia de datos entre ambos dispositivos (Chu, 2012). Este tipo de dispositivos ofrecen características que simplifican el trabajo con el hardware reconfigurable. En primer lugar, es posible conectar nuestra arquitectura con dispositivos mapeados en memoria dentro del microprocesador, de modo que, desde un sistema operativo se pueden ejecutar programas que lean o escriban directamente en las direcciones de memoria vinculadas al FPGA. Esto abre la posibilidad de controlar el dispositivo de hardware desde el software, lo cual facilita el uso de la arquitectura y la vuelve más flexible. En el presente trabajo presentamos un acelerador de hardware, es decir, un circuito dedicado cuyo fin es la implementación de un algoritmo a mayor velocidad que el software, para la implementación de simulaciones de dinámica de redes Booleanas. El acelerador está conectado directamente con un microprocesador embebido en un SoC, desde el cual se ejecuta un sistema operativo que permite el control del acelerador mediante software que lo configura a bajo nivel.

3.2 Módulo de simulación de dinámica de la red implementado en este trabajo

Se diseñó un módulo de hardware que ejecuta la simulación de la red a partir de un estado inicial, similar al propuesto en el trabajo de Ferreira y Vendramini (2010), que integra un *datapath* y una unidad de control formada por una FSM.

El *datapath* está conformado por dos submódulos: el primero se encarga de cargar los estados iniciales y almacenar el estado actual de la red, mientras que el segundo integra a los componentes que realizan las funciones de los nodos. Este último submódulo llama internamente a los componentes que implementan en paralelo las funciones de cada nodo individual y conecta sus puertos de manera congruente con la topología de la red. Al manejar de manera interna la interconexión de los nodos, se les puede tratar como a una unidad con una sola entrada y una sola salida de n bits, tal que n es el número total de nodos (Figura 15). Esto simplifica la adaptación de la arquitectura para redes diferentes, puesto que, si se quisiera implementar otra red de n nodos, el submódulo de interconexión sería el único componente en todo el diseño que tendría que ser cambiado.

Este enfoque de implementación de las funciones de cada uno de los nodos síncronos individuales que operan en paralelo permite reducir la complejidad algorítmica. De manera informal, la complejidad algorítmica representa el número de pasos que toma resolver un problema y generalmente esto se puede escribir mediante una función matemática. Generalmente, se utiliza una cota superior de tal función para describir la complejidad de un algoritmo. Por ejemplo, si el número total de pasos de un algoritmo está dado por la función $f(n)=n^3+2n^2+1$, donde n es el tamaño del problema (*e.g.*, número de nodos) entonces su cota superior se encuentra descrita por la función $O(n^3)$, dado que n^3 es el término de la función que crece más rápido (Sedgewick y Flajolet, 2013). En este trabajo, complejidad se usará solo en términos de tiempo de cómputo, tomando como unidades el número de ciclos de reloj que tiene que realizar el hardware. La complejidad algorítmica de actualizar el estado de la red es $O(1)$, es decir, solo se requiere un paso para realizar la actualización de todos los nodos, lo que se traduce a un solo ciclo de reloj para actualizar todo el sistema. Esto representa una ventaja sobre una implementación secuencial, dado que en esta la actualización del sistema requiere de visitar cada nodo y sus vértices adyacentes, lo que genera una complejidad de $O(n+e)$, donde n es el número de nodos y e es el número de aristas en la red (Ferreira y Vendramini, 2010).

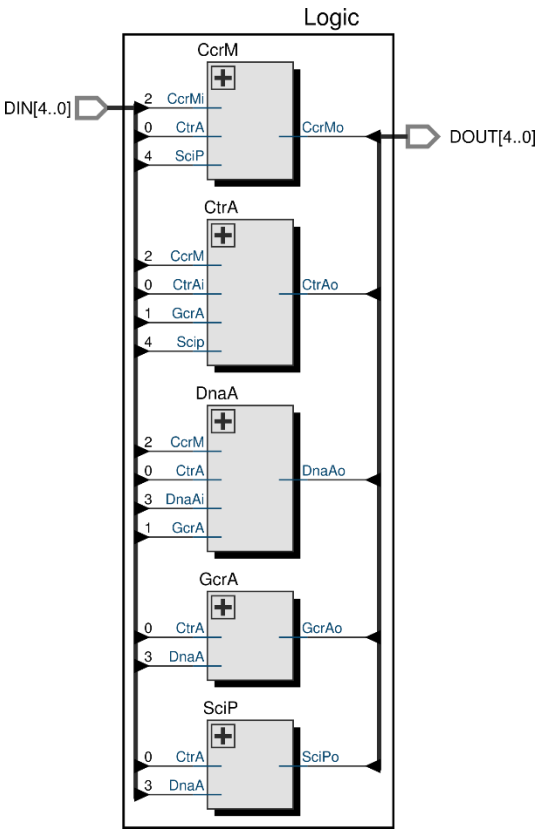

Figura 15. Submódulo de procesamiento. Se muestra la implementación del submódulo de procesamiento de datos para la red de cinco nodos asociada al ciclo celular de *C. crescentus*. Este componente posee salida de cinco bits, provenientes de las salidas de los módulos de procesamiento y equivale al valor del nodo en el estado actual de la red. Adicionalmente, el componente tiene una entrada de cinco bits, conectados de forma individual con los módulos de procesamiento correspondientes.

El submódulo de almacenamiento consta de un registro de *n flip-flops*, cada uno de los cuales almacena el estado de un nodo. Cada *flip-flop* se conecta con un dispositivo que permite cargar el estado inicial del nodo, así como establecer si se realizan perturbaciones sobre este. Para realizar estas operaciones se utilizan componentes llamados multiplexores, cuya salida es seleccionada de entre dos entradas según el valor de un bit de selección (Figura 16).

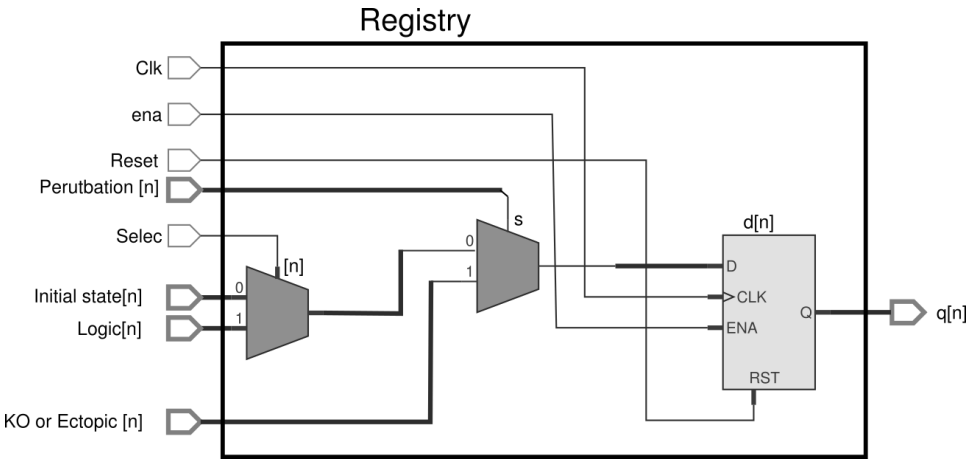

Figura 16. Módulo de almacenamiento con selección de entrada. El estado actual de la red es guardado en un registro de *n flip-flops* con *enable* y *reset*. La entrada de datos a este registro es controlada por dos multiplexores. El primer multiplexor es controlado por una señal llamada *"Selec"*, y tiene como propósito decidir si el valor que pasará al *flip-flop* en el siguiente ciclo de reloj es el estado inicial, o el estado que venga del submódulo de procesamiento. El segundo *flip-flop* es controlado por la señal *"Perturbation"*, y su función es seleccionar si el valor que pasa al *flip-flop* es el que viene del multiplexor anterior, o si es el valor de una entrada denominada *"KO or Ectopic"*, que se mantiene fijo durante toda la simulación. Si *"Perturbation"* vale 0, el *flip-flop* carga el estado del primer multiplexor. De lo contrario, se carga el valor de *"KO or Ectopic"* cuyo valor determina si la perturbación es un noqueo, *"KO or Ectopic"*=, o si se trata de expresión ectópica *"KO or Ectopic"*=1.

La FSM que conforma la unidad de control se encarga de que manejar las señales que controlan al registro, de manera tal que en el estado inicial se cargue al inicio de la simulación y luego se carguen los estados que provienen del submódulo de procesamiento. Adicionalmente, esta recibe una señal llamada "Found" que viene del módulo de búsqueda de atractores, la cual le dice en qué momento detener la simulación. La figura 17 muestra el diseño integrado del módulo de simulación.

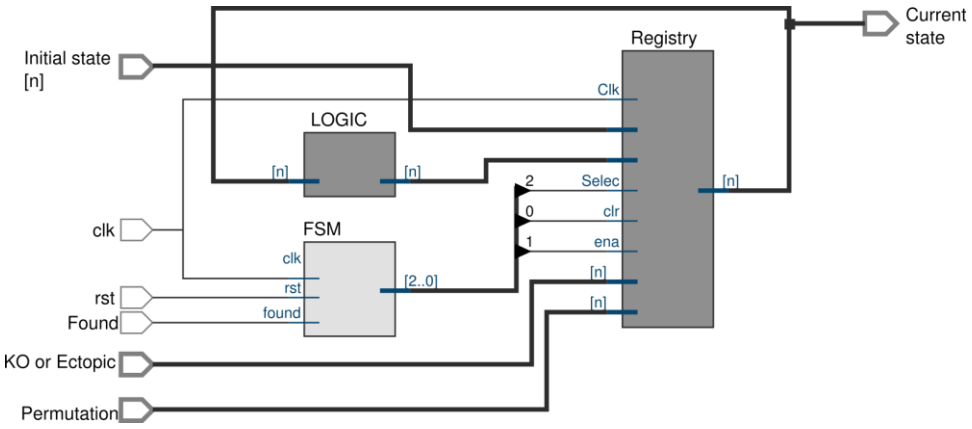

Figura 17. Módulo de simulación de red. Este circuito integra al *datapath* y la unidad de control descritas en esta sección. El módulo entrega a la salida el estado actual de la simulación.

3.3 Módulo de búsqueda de atractores

Para la búsqueda de atractores en la dinámica de la red, se diseñó un módulo que implementa el algoritmo propuesto por Bhattacharjya y Liang (1996). Este algoritmo se presta particularmente para una implementación en hardware, porque a diferencia de otros algoritmos (Irons, 2006; Zhang *et al*., 2007) no requiere recuperar datos de la memoria, pero demanda la implementación en paralelo de dos simulaciones de la dinámica. De manera teórica, el algoritmo funciona de la siguiente manera: Tenemos dos simulaciones a las cuales llamaremos S y S_{2x}. Estas simulaciones cumplen la condición de que ambas parten del mismo estado inicial, pero la simulación S_{2x} se ejecuta al doble de velocidad que la simulación S. Los de la red en cada simulación en un tiempo t son representados como $S(t)$ $S_{2x}(t)$. Debido a que el número de estados que puede tener el sistema es finito (aunque enorme), y que la trayectoria de la dinámica es determinista, eventualmente ambas simulaciones convergen en un mismo estado en un tiempo t_j, de modo que $S(t_j) = S_{2x}(t_j)$. Una vez alcanzado este punto, la simulación S_{2x} se detiene, mientras que S se ejecuta por P iteraciones más, donde P es el número de iteraciones que le tome volver a converger en un mismo estado con la simulación S_{2x}. Partiendo de un estado

inicial *i*, el número P_i de iteraciones por las que pasa la simulación *S* antes de que se cumpla la condición $S(t_j+P_i) = S_{2x}(t_j)$ equivalen al periodo del atractor al que converge la red partiendo del estado *i*, y los P_i estados que adopta la red durante este periodo conforman el atractor correspondiente. La complejidad del algoritmo es $O((P+T)$ * "actualización de los nodos"), donde *T* el número de iteraciones que tuvo que realizar el sistema antes de llegar al atractor. Como mencionamos anteriormente, en un enfoque secuencial la complejidad algorítmica de la actualización de todos los nodos es de $O(n+e)$, por lo que la complejidad de la búsqueda de un atractor es de $O((P+T) * (n+e))$. Sin embargo, dado que en la implementación aquí descrita la actualización de los nodos tiene una complejidad de $O(1)$, la complejidad de la búsqueda de un atractor es de $O(P+T)$. La figura 18 muestra un ejemplo de la implementación de este algoritmo para la búsqueda de un atractor hipotético.

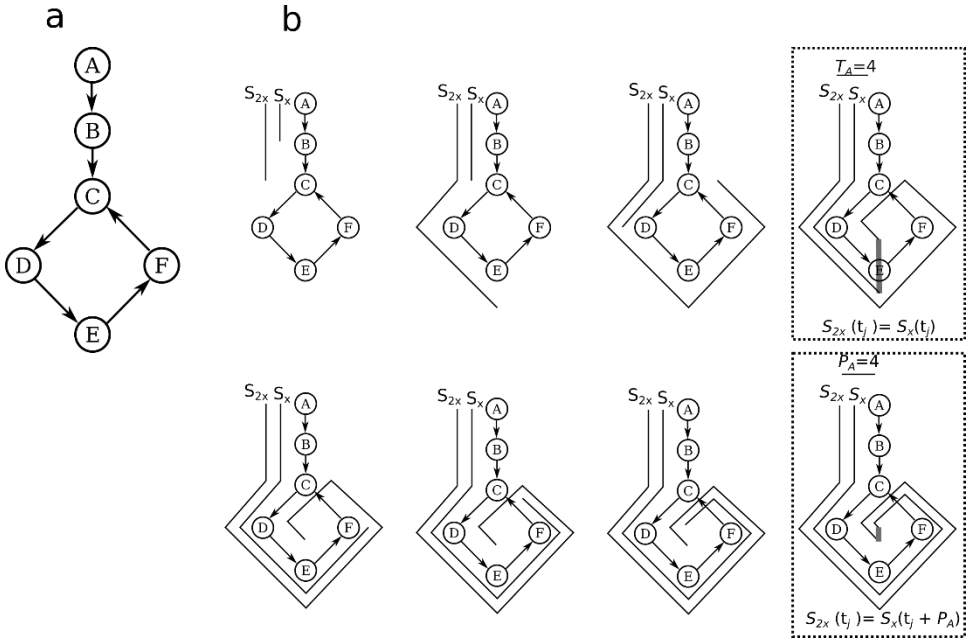

Figura 18. Algoritmo de búsqueda de atractores. a) Se presenta una trayectoria hipotética de la dinámica de una red a partir del estado inicial "A". La dinámica presenta una cuenca de atracción formada por dos estados, "A" y "B", y un atractor de periodo 4 formado por los estados "C", "D", "E", y "F". b) Implementación del algoritmo de Bhattacharjya y Liang (1996) sobre la trayectoria descrita en el recuadro a. Las simulaciones, *S* y S_{2x}, avanzan a lo largo de la trayectoria en iteraciones discretas, en las cuales, S_{2x} avanza dos estados por iteración, mientras que S solo avanza uno. Cuando se cumple la condición $S(t_j) = S_{2x}(t_j)$, S2x se detiene y la simulación S continúa avanzando hasta que se cumpla la condición $S(t_j + P_A) = S_{2x}(t_j)$, donde P_A es el periodo del atractor al que llega la trayectoria a partir del estado inicial A. El número de Iteraciones totales es igual al número de iteraciones transitorias, T_A, por las que tiene que pasar la simulación *S* antes de coincidir con S_{2x}, más P_A iteraciones por las que transita la red hasta que ambas simulaciones coincidan de nuevo.

A groso modo, la implementación en hardware de este algoritmo se realizó de la siguiente manera: se diseñó un componente que incorpora a dos copias del submódulo de simulación de red descrito en la sección 3.2 y un comparador que determina cuando ambas redes convergen en sus estados.

Para obtener dos simulaciones que se ejecuten en paralelo y cumplan la condición de que una se ejecute al doble de velocidad de la otra, como corresponde a *S* y S_{2x}, se incluyó en el diseño a dos copias del módulo de la sección 3.1, y se le conectó una a un reloj que oscila al doble de velocidad de la otra. Específicamente, se utilizó un reloj con frecuencia de 25MHz para la copia que corresponde a *S*, y de 50MHz para la que corresponde a S_{2x}. El estado de la red en cada

una de las simulaciones es entregado a la salida del componente, y pasa a un módulo comparador que verifica en tiempo real si ambas salidas son iguales, de manera que el sistema pueda saber cuándo se cumplen las condiciones $S(t_j) = S_{2x}(t_j)$ y $S(t_0+P_i) = S_{2x}(t_j)$. Para asegurarnos que la simulación S_{2x} se detenga a tiempo, la FSM de la copia con reloj rápido se detiene la primera vez que se detecte una coincidencia, mientras que la de la otra copia se detiene en la segunda coincidencia. Adicionalmente, para indicar que ya se concluyó la búsqueda de atractores, la máquina de estados con reloj de lento activa una señal llamada *"DONE"* al momento de recibir por segunda vez la señal del comparador (Figura 19).

Para determinar si ambos estados son iguales, los pares de bits que representan al estado de un mismo nodo en las salidas de cada submódulo de simulación pasan a través de una función (XNOR). Esta función se utiliza porque sólo entrega un valor verdadero cuando ambas entradas son iguales, como se muestra en la tabla 4.

S	***S*$_{2x}$**	***S* XNOR *S*$_{2x}$**
0	0	1
0	1	0
1	0	0
1	1	1

Tabla 4. Tabla de verdad de la función XNOR.

Cada uno de los *n* bits individuales, *b,* de salida de la función anterior son alimentados a la función q = b_0 AND b_1 AND b_2 … AND b_n, cuya salida sólo es igual a "1" cuando todos los bits de salida de la función XNOR valgan "1", lo cual asegura que solo se entregará un valor verdadero cuando ambas simulaciones converjan en el mismo estado de red. El resultado de esta función es exportado a la salida del comparador, la cual es alimentada en la entrada "*Found*" de las FSM que controlan a las unidades de procesamiento de cada copia del submódulo de simulación.

3.4 Módulo de registro de resultados

Para reportar los periodos de los atractores, así como una representación numérica de su trayectoria, la arquitectura presentada en este trabajo emplea contadores en hardware que registran los valores del módulo comparador y de los estados de la simulación. En primer lugar, se implementa un contador que registra el número de ciclos que da el reloj de 25MHz entre la primera y la segunda vez que el comparador entrega un “1”, lo cual corresponde al valor de P_i. En segundo lugar, tenemos un contador que guarda el número de ciclos del reloj de 50MHz que transcurren hasta encontrar el atractor, lo cual sirve para estimar el tiempo que le tomó al dispositivo completar la simulación. Finalmente, se tiene un grupo de *n* contadores que registran el número de veces que estuvo prendido cada gen durante el periodo del atractor. Al dividir el número de veces que un nodo estuvo prendido entre el periodo del atractor, se puede conocer la frecuencia de activación del nodo en un atractor. El presentar las frecuencias de activación de los nodos arregladas como un vector es una manera corta de reportar la dinámica de un atractor (Irurzun-Arana *et al.,* 2016). De hecho, esta es una representación que resulta práctica para el propósito de realizar estudios de perturbación de red, como se hace en el trabajo

de Lu y colaboradores (2014). La figura 19 muestra la integración de los contadores con los módulos descritos en las secciones anteriores, correspondiendo al diseño del acelerador de hardware completo.

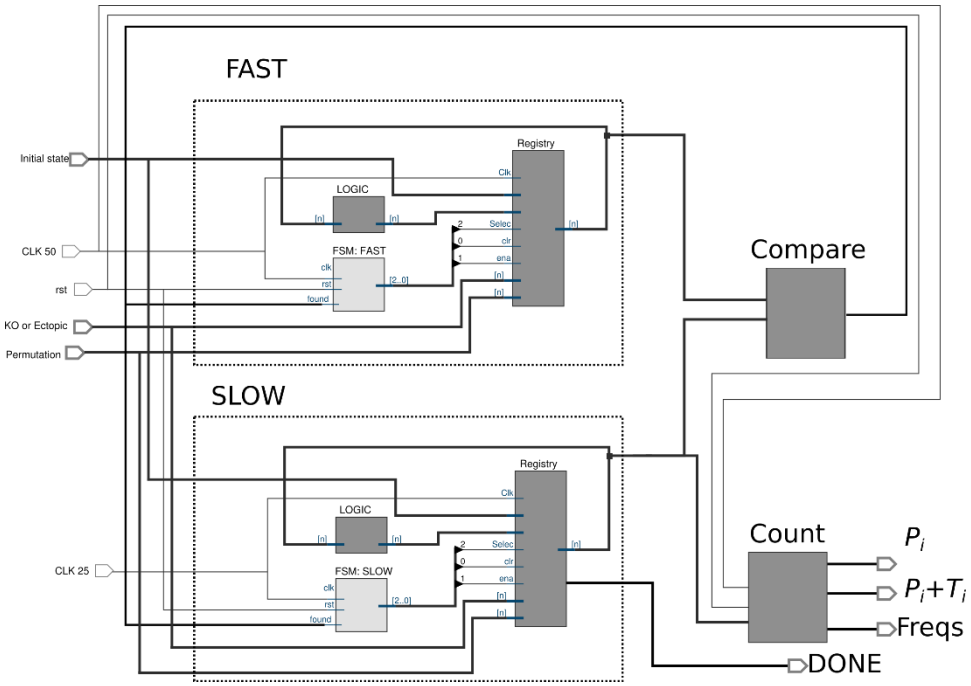

Figura 19. Acelerador de hardware. El dispositivo final integra las dos copias del módulo de simulación, el módulo de comparación, y los contadores. Esta arquitectura permite la búsqueda automática de un atractor a partir de un estado inicial. Las entradas y las salidas de este dispositivo se vinculan a regiones mapeadas en memoria en el microprocesador, el cual envía los parámetros para la simulación.

3.5 Módulo de interconexión con microprocesador.

Para intercambiar datos con el FPGA de manera práctica, es necesario conectar la arquitectura con un microprocesador de propósito general que permita enlazar el diseño con una capa de software. Para esto se implementaron en el FPGA módulos de interconexión mediante el protocolo AVALON®. Este protocolo es un estándar que define el tipo de conexiones para la comunicación entre un FPGA y un microprocesador embebido, así como las especificaciones del flujo de datos entre ambos dispositivos. Los archivos de configuración para enlazar la arquitectura en FPGA con direcciones específicas de la memoria física del microprocesador fueron generados con la herramienta de diseño Qsys, la cual forma parte de la plataforma de EDA Quartus II v16 (Intel). De manera específica se asignaron cuatro direcciones de memoria que el microprocesador puede usar para enviar al FPGA el estado inicial, la señal de activación de perturbaciones, el tipo de perturbaciones y la señal de *reset*. De la misma manera, se asignaron direcciones de memoria sobre las que el FPGA puede guardar el valor del contador de periodo, los contadores individuales de cada gen, la señal que marca el final de la simulación y el número total de ciclos de reloj.

3.6 Síntesis y generación de binario de configuración.

Las descripciones de hardware se realizaron en el lenguaje VHDL y fueron implementadas en la tarjeta DE-1 (Terasic), la cual contiene el chip de Cyclone V SoC 5CSEMA5F31C6 (Intel FPGA). Este chip integra en un solo dispositivo a un FPGA de 85K elementos lógicos, y a un microprocesador de doble núcleo llamado ARM Cortex-A9, el cual fue utilizado para el control del sistema desde software. Las restricciones de los relojes y otros detalles que son particulares al FPGA con el que se trabajó. fueron incorporadas mediante scripts en lenguaje TCL (*Tool Command Language*). La dinámica de operación de los componentes individuales fue depurada mediante el programa ModelSim (Altera corporation), y la síntesis de la arquitectura se realizó mediante la plataforma de EDA Quartus II v16 (Intel). El binario de configuración para la arquitectura se generó en formato ".sof". Las configuraciones en este formato pueden ser cargadas al FPGA por el ARM desde el arranque de un sistema operativo.

3.7 Generación de distribuciones de Linux

Se generaron desde código fuente dos distribuciones de Linux que permiten la configuración automática del FPGA, y la comunicación del FPGA mediante una capa de software.

Se desarrolló un sistema de arranque que permita realizar la configuración del hardware al inicio del sistema operativo,

3.7.1 Generación de sistema de arranque

De manera muy simple, el arranque de un sistema operativo comienza con la ejecución de una secuencia de funciones de bajo nivel específicas al microprocesador, en las que se habilitan los componentes de hardware de manera ordenada (Figura 16). Primero se ejecutan una serie de directivas que vienen programadas de fábrica en la ROM, las cuales ejecutan desde el disco duro o memoria externa a un programa de precarga ("pre*Loader*"), también conocido como Programa Cargador Secundario (SPL por sus siglas en inglés Secondary Program *Loader*). El programa de precarga se encarga de configurar el microprocesador a bajo nivel, para asegurarse de que el cargador de arranque del sistema operativo pueda ser ejecutado. Debido a que se está modificando las configuraciones de hardware en la porción de FPGA del dispositivo, es necesario generar un SPL específico para cada una de las arquitecturas implementadas en este trabajo.

El cargador de arranque U-Boot, ofrece la facilidad de que el código fuente de su programa de precarga puede ser modificado para generar programas de precarga específicos. La porción del código fuente necesario para obtener un SPL basado en U-Boot específico para nuestra arquitectura se generó de manera automática mediante el programa BSP Editor® (Intel), a partir de los archivos de configuración generados durante la síntesis de la arquitectura.

Para arrancar el sistema operativo, se utilizó el cargador de arranque U-Boot, modificando parte del código fuente especificando que antes de arrancar el sistema operativo, configure al FPGA con los archivos de configuración generados en el proceso de síntesis. La compilación del cargador de arranque, así como la de todos los demás componentes de software, se realizó con la cadena de herramientas de compilación cruzada gcc-linaro-arm-linux-gnueabihf-4.9-2, puesto que el microprocesador en el que se ejecutará el sistema operativo tiene una arquitectura tipo ARM.

3.7.2 Implementación de Linux

Se compiló una versión del *Kernel* 4.1 de Linux, la cual ha sido modificada por Intel® para su uso en sistemas embebidos. Mediante la herramienta de compilación de kernel menuconfig se configuraron los parámetros específicos para el microprocesador específico usado en este proyecto.

Dado que nuestro acelerador de hardware se conecta físicamente a ciertas direcciones de memoria del microprocesador, fue necesario generar un árbol de dispositivos específico para nuestro diseño. Los árboles de dispositivos son estructuras de datos que determina las direcciones de los componentes de hardware en un dispositivo, de modo que puedan ser accedidos por el *Kernel*, además, especifican los anchos de memoria física para cada dispositivo y los drivers que deben de ser cargados para controlarlo. Mediante el programa sopc2dts (Altera®) se generó el código fuente para el árbol de dispositivos a partir de los archivos de configuración generados en Qsys™(Intel®). El código fuente se compiló en un árbol de dispositivos mediante el programa DTC (*Device Tree Compiler*).

El sistema de archivos del sistema operativo, así como los archivos de sistema que no estuvieran directamente relacionados con el acelerador de hardware, se generaron mediante las herramientas Buildroot y Busybox.

Una vez compilados todos los componentes se generó una imagen de disco mediante el programa disk dump, la cual fue particionada con la fdisk. Sobre esta imagen se montaron en las particiones adecuadas los archivos de sistema operativo y configuración de hardware para la implementación de cada uno de los dos aceleradores desarrollados en este trabajo.

3.8 Control por software del acelerador de hardware

Para poder controlar diseño de hardware desde Software, se elaboraron encabezados en C que se declaran las funciones para comunicarse con el FPGA y se definen las variables que afectan esa comunicación, como las direcciones de memoria en las que se tiene que leer y escribir la transferencia de datos, el ancho de las direcciones, etc. Adicionalmente, se desarrolló un programa en C que automatiza la búsqueda de atractores en Hardware, el cual incluye a los encabezados diseñados para el control del acelerador, e implementa una rutina para la simulación exhaustiva de todos los estados posibles.

En una fase inicial, el programa escribe un “1” en la dirección de memoria vinculada a la señal de *reset*. Con esto se asegura de que los contadores estén en cero, y las FSMs estén en su estado inicial. Después, carga un estado inicial en la dirección de memoria designada para la carga de estados. Dado que *reset* está configurado en “1”, la simulación no arranca, aunque ya esté cargado el estado inicial. Luego sobrescribe la dirección de memoria del *reset* para configurarla en “0”, iniciando así la simulación. El programa entonces espera hasta que el FPGA escriba un “1” en la dirección de memoria que indica la conclusión de la búsqueda de atractores. Cuando recibe la señal de que terminó la búsqueda, el programa lee el valor en las direcciones de memoria vinculadas a los resultados del FPGA. El valor en la dirección de memoria del contador de ciclos de reloj la guarda como una variable entera sin signo. Los valores del periodo y la frecuencia de activación son guardados en un vector, el cual es agregado a un arreglo. En la siguiente fase, repite el mismo proceso para todos los estados en el espacio, hasta terminar la simulación. Para cada iteración, el programa verifica si el vector que describe al periodo y

la frecuencia del nuevo atractor es igual a uno de los que ya están guardados en el arreglo; si no hay otro vector igual, el vector es agregado al arreglo, en caso contrario, su valor es ignorado y se prosigue con la simulación. El valor en la dirección de memoria del contador es sumado a la variable definida en la fase inicial, de manera que se vaya actualizando en cada simulación. Al final, el programa imprime en pantalla el arreglo y el número de ciclos de reloj.

Para realizar un análisis de perturbación, se creó un programa que realiza el mismo proceso que el programa de exploración exhaustiva, solo que lo realiza *n* veces, simulando la eliminación o noqueo de un gen en cada iteración del proceso de exploración.

3.9 Evaluación comparativa de la arquitectura de hardware

Se realizó una evaluación comparativa de la simulación de las dos redes Booleanas de las redes asociadas al ciclo celular de *C. crescentus* (Quiñones-Valles *et al*., 2014) y la red de formación de tumores en CAC (Lu *et al*., 2015), las cuales fueron descritas respectivamente en las secciones 1.2.1 y 12.3 del presente trabajo. El desempeño de la arquitectura de hardware presentada en este trabajo se comparó contra dos implementaciones en R realizadas mediante los paquetes SPIDDOR (Irurzun-Arana *et al.*, 2016) y BooleNet (Müsel *et al.*, 2010).

Las implementaciones de software se ejecutaron en un servidor Dell Precision T7600, con un procesador Intel Xeon® CPU-E5-2620 de 6 núcleos con frecuencia de reloj 2.0GHz, 16 Gb de memoria RAM DDR3, con el sistema operativo Windows 7.

El tiempo total en segundos que tomó implementar la dinámica de cada red para los aceleradores de hardware fue estimado mediante el comando de Linux “time” ejecutado en el microprocesador embebido en el chip del FPGA, mientras que los tiempos de las implementaciones en software se estimaron mediante la función de R “system.time”.

IV Resultados

Se obtuvieron dos distribuciones de Linux, las cuales ejecutan en el arranque la configuración del FPGA para los aceleradores de hardware para la búsqueda de atractores en las redes usadas como casos de estudio.

En la evaluación comparativa para la red asociada al ciclo celular de *C. crescentus*, los tiempos de simulación en todas las plataformas fueron inferiores a 0.01s, de modo que no se considera que haya una ventaja significativa de una implementación sobre la otra. En contraste, para la red asociada a la formación de tumores en CAC hubo diferencias de por lo menos un orden de magnitud en los tiempos de ejecución entre las distintas implementaciones, como se ilustra en la tabla 5.

Simulación de la dinámica de la red asociada a la formación de tumores			
Plataforma	Tiempo real (s)	Ciclos de reloj	Tiempo FPGA (ms)
Hardware	36.51	20307631	406.2
BooleNet	416.34	NA	NA
SPIDDOR	33202.11	NA	NA

Tabla 5. Evaluación comparativa de los tiempos de ejecución. Para cada una de las implementaciones se reporta el tiempo real que tomó concluir la simulación. En el caso particular de la implementación en hardware, se reporta adicionalmente el número de ciclos de reloj que le tomó al FPGA realizar la simulación y la conversión de esos ciclos de reloj a milisegundos, tomando en cuenta que el reloj tiene una frecuencia de 50MHz. La diferencia entre el tiempo real y el tiempo de simulación en el FPGA se debe al retardo ocasionado por la capa de software en el microprocesador. NA= No Aplicable.

La implementación en hardware fue 11.4 veces más rápida que la implementación con BooleNet, y 909.3 veces más rápida que la implementación con SPIDDOR. La diferencia drástica entre los tiempos de ejecución entre los programas se origina porque SPIDDOR busca atractores mediante la exploración exhaustiva del espacio de estados, mientras que BooleNet acelera el proceso mediante el algoritmo de satisfacción propuesto por Dubrova y Teslenko (2009), descrito en la sección 1.1.5 del presente trabajo. La implementación en hardware presentada en este trabajo realiza búsqueda de atractores mediante una búsqueda exhaustiva, pero obtiene un mejor desempeño debido al diseño en paralelo de la arquitectura.

En el trabajo original de Lu y colaboradores (2015) para la red de 21 nodos no se reportan las trayectorias de los atractores, pero si se presenta el número de los atractores encontrados y el periodo de cada uno de estos. Adicionalmente se enumera cuántos atractores se encuentran en las categorías de *Proliferation*, *Resting*, y *Limited proliferation*, que como se mencionó en la sección 1.2.3 corresponden respectivamente a atractores en los que el nodo de proliferación se encuentra encendido todo el tiempo, apagado todo el tiempo, y prendido durante una porción del tiempo. Cabe aclarar que no reportan las frecuencias específicas de prendido del nodo de proliferación en los atractores de *Limited proliferation*¸ por lo que no es posible una comparación directa

Para comparar nuestros resultados con los reportados por (Lu *et al.*,2015) se clasificó a los atractores según las categorías de *Proliferation*, *Resting*, y *Limited proliferation*, y se verificó que las proporciones fuesen las mismas. Tanto SPIDDOR como nuestra implementación en hardware encontraron 2 atractores en *Proliferation*, 18 atractores en *Resting* y 8 atractores en *Limited proliferation*, dando un total de 28 atractores, coincidiendo con el total y las proporciones que reportan por Lu y colaboradores (2015). La implementación basada en satisfacción Booleana implementada en BooleNet, solamente encuentra a 12 atractores, de los cuales 2 se clasificaron dentro de *Proliferation*, 6 en *Resting* y 4 en *Limited proliferation*, sin incluir ningún atractor que no estuviese contenido en los 28 atractores encontrados mediante búsqueda exhaustiva. Esta diferencia posiblemente se deba a que a final de cuentas los algoritmos de satisfacción son un método heurístico, y por lo tanto no siempre encuentran todos los atractores posibles. No se analizan las frecuencias de demás nodos, porque no hay forma de compararlos con los resultados de (Lu *et al.*, 2015) y la interpretación de sus dinámicas va más allá del alcance del presente proyecto.

La simulación de perturbación de nodos se completó en un tiempo real de 774.3771 segundos, con un total de 426,460,293 ciclos de reloj en el FPGA, lo cual es equivalente a aproximadamente 8.52 segundos.

V Discusión

Por un lado, para la simulación de una red de cinco nodos, nuestra implementación en hardware no presentó una ventaja perceptible sobre las implementaciones en software. Por lo que consideramos que, a esta escala del número de nodos componentes de una red, es mejor utilizar implementaciones de software.

Por otro lado, para la simulación de la dinámica de una red de 21 nodos, la implementación en hardware desarrollada en este trabajo presenta un mejor desempeño que implementaciones de software publicadas en la literatura. El tamaño de esta red se encuentra casi en el límite de lo que actualmente es posible mediante software para la búsqueda exhaustiva de atractores. Es particularmente notable que se superó el desempeño incluso de la simulación realizada en el programa BooleNet, la cual empleó el método heurístico del algoritmo de Dubrova y Teslenko (2011). A pesar de que solo se está trabajando con un caso de estudio, consideramos que la diferencia en el desempeño es suficiente para postular que las simulaciones en hardware son una alternativa viable para la simulación de redes de mediano a gran tamaño, como son todas las redes reales biológicas. Los incrementos en la velocidad de simulación son congruentes con los resultados presentados por trabajos previos en la implementación de redes Booleanas en FPGAs (Ferreira y Vendramini, 2010; Zerarka *et al*., 2004).

En la actualidad existen implementaciones en software de algoritmos de satisfacción Booleana descritos en la sección 1.2.3, las cuales logran realizar en un tiempo razonable la simulación de la dinámica de redes de hasta miles de nodos. Sin embargo, éste enfoque pierde su eficacia conforme aumenta el promedio del grado de interconectividad, *k*, de los nodos para una red (Veliz-Cuba *et al*., 2014). La ventaja de la arquitectura aquí descrita es que, como la actualización de todos los nodos de la red se realiza de manera paralela, el tiempo de simulación de la red no se ve afectado por el grado de interconectividad de la misma.

Una posible objeción a la relevancia de la independencia al grado de interconexión que presenta la implementación aquí descrita es que en las redes biológicas reales los nodos tienden a presentar valores de *k* cercanos a 2 (Balleza *et al*., 2008). Este dato indica que los grados de interconectividad son manejables por los algoritmos tipo SAT (Veliz-Cuba *et al*., 2014; Tamura y Akutsu, 2009; Devloo *et al.*, 2003). Sin embargo, estudios recientes basados en ChiP-Seq y RNA-Seq han revelado una amplia diversidad de sitios de unión a DNA previamente desconocidos, apuntando a que muy posiblemente las redes de regulación están mucho más interconectadas de lo que se cree actualmente (Minch *et al*., 2015; Galagan *et al*., 2013). El estudio de redes con mayor interconectividad demanda el uso de metodologías de búsqueda de atractores que no se vean afectadas por el grado de conectividad de la red, tal y como la que se presenta en este trabajo.

Desde el punto de vista electrónico, el presente trabajo contrasta con otras implementaciones en hardware reportadas anteriormente en cuatro aspectos importantes que contribuyen hacia la incorporación de los FPGAs como una plataforma de cómputo de uso generalizado en el estudio de redes biológicas.

En primer lugar, a diferencia de los trabajos de Ferreira y Vendramini (2010) y Zerarka *et al.*, (2004), en el presente trabajo se implementa la búsqueda de atractores de redes Booleanas que representan a redes biológicas reales en lugar de redes aleatorias generadas en la computadora. Aunque es cierto que en los artículos de Miskov-Zivanov y colaboradores (2011a; 2011b) ya se habían simulado redes reales en FPGAs; sin embargo, la arquitectura que ellos reportan no permite la búsqueda automática de atractores a partir de un estado inicial, de hecho, su

implementación no permite siquiera la entrada y salida de datos sin necesidad de interacción física del usuario con el hardware.

En segundo lugar, las arquitecturas de hardware presentadas, tanto por Ferreira y Vendramini como por Zerarka y colaboradores, solamente entregan el periodo de los atractores encontrados, pero no permiten obtener ninguna otra información acerca de estos. Al incorporar en nuestro diseño el almacenamiento de los atractores, aun cuando sea solo como frecuencias de encendido para cada uno de los nodos, se puede aspirar a un mayor entendimiento de la dinámica de las redes que representan. Las posibilidades que se abren al incorporar el almacenamiento de los atractores como frecuencias de encendido se hacen más claras al considerar la tercera diferencia de la implementación desarrollada en presente trabajo con otras publicadas anteriormente: la incorporación de la perturbación de nodos.

Como se mencionó en la sección 1.3.2, en el trabajo de Lu y colaboradores (2015), se utilizó un análisis de perturbaciones para predecir posibles blancos moleculares a partir de la dinámica de la red. Siendo más específicos, lo que hicieron fue comparar las frecuencias de activación de los nodos que llevan hacia la apoptosis en la red sin perturbar, con las frecuencias de estos mismos genes ante la presencia de una perturbación, y tomaron las perturbaciones que más aumentaran esta frecuencia como los mejores candidatos a ser blancos moleculares. Al incorporar la posibilidad de perturbar genes aún después de haber sintetizado la arquitectura, con una manera de obtener las frecuencias de activación de los nodos, abrimos la puerta para realizar estudios de perturbación en hardware como los que realizaron en software Lu y colaboradores (2015).

Nuestro trabajo difiere además de las implementaciones mencionadas anteriormente, en no limitarnos a presentar solamente el tiempo promedio que toma encontrar un estado inicial, sino que a través del software llegamos a la implementación automática de la búsqueda exhaustiva de atractores a partir de todos los estados iniciales posibles, acercándonos más al modo de trabajo que ofrecen las plataformas puramente en software.

Aunque como tal este trabajo no presenta ningún hallazgo biológico nuevo, nuestro objetivo era desarrollar y evaluar una metodología de alto desempeño para acceder al estudio de redes de regulación de tamaños grandes, que hasta la fecha no se pueden abordar sin recurrir a técnicas heurísticas debido al elevado tiempo de cómputo requerido, el cual se encuentra en el orden de meses o hasta años. Las implicaciones de nuestro enfoque se abordan a mayor profundidad en la sección de perspectivas.

VI Perspectivas

El presente trabajo deja muchas avenidas para expansiones y mejoras tanto en términos del uso de hardware reconfigurable como herramienta para obtener conocimiento biológico relevantes, como desde el punto técnico de mejorar y expandir la arquitectura de hardware.

Desde el punto de vista del uso de FPGAs como una plataforma para análisis de redes biológicas, valdría la pena comparar de manera más sistemática los tiempos de simulación de una arquitectura de cómputo en hardware, con los tiempos de simulación de varios programas que utilicen diferentes algoritmos. Además, se tendría que considerar a todo un espectro de combinaciones de tamaños de red y grados de interconexión promedio. También sería interesante implementar alguna de las heurísticas en un FPGA, para intentar acceder a la dinámica de redes aún mayores.

Desde el punto de vista electrónico, queda abierta la posibilidad de aumentar el grado de paralelismo, ya sea mediante la implementación de múltiples copias del diseño en hardware, o incluso mediante un clúster de FPGAs, que permitan realizar en paralelo la implementación de la dinámica de una red a partir de múltiples estados iniciales. Otra posibilidad técnica interesante es adaptar el diseño para implementarlo en un FPGA con un reloj de mayor frecuencia, de modo que sea posible aumentar todavía más la ventaja en velocidad de una implementación en comparación con las simulaciones en software.

Para llevar este enfoque hacia un uso generalizado entre la comunidad de biólogos, también hace falta desarrollar una arquitectura de hardware que sea totalmente reconfigurable desde software. Esto permitiría al usuario generar redes con reglas y topologías personalizadas sin la necesidad de tener que diseñar y compilar una implementación en hardware. Durante el desarrollo de este proyecto se planteó lograr justamente esta meta mediante el diseño de una arquitectura reconfigurable que se sobrepone a la capa física del FPGA. Esta arquitectura pudiera ser controlada desde un microprocesador. También se planteó el diseño de un elemento lógico superpuesto para implementar los módulos que ejecutan las funciones de los nodos, y establecer una red de interconexión reconfigurable como la implementada en el trabajo de Ferreira y Vendramini (2010). Asimismo, se planteó crear una interfaz de usuario gráfica que permitiera al usuario utilizar estos módulos sin necesidad de entender el funcionamiento del hardware. Se logró de manera exitosa el implementar el módulo de nodo reconfigurable, y se logró implementar un programa en Java que permite convertir un archivo de una red en formato SBML a una serie de tablas de verdad para cada gen. Desafortunadamente, el reto de lograr una interconexión reconfigurable resulto ser demasiado grande para implementarse en el periodo establecido para este proyecto, así que se optó por implementar una red fija. Se pretende concluir esta meta en un trabajo posterior.

Finalmente, como nuestro esquema búsqueda de atractores no se basa en una heurística sino en un método exhaustivo, se obtienen resultados congruentes con el modelo de la red sin incurrir en omisiones o resultados espurios. Por este motivo, consideramos que el enfoque de aceleración por hardware puede ser aplicado para generar en un tiempo razonable predicciones congruentes con los modelos de una red construidos a partir de información biológica. Esto se ejemplifica en el contraste entre los resultados de la simulación de la red asociada a formación de tumores en CAC (Lu *et al.*, 2015) obtenidos mediante un método de búsqueda exhaustivo los resultados obtenidos mediante un método heurístico. La congruencia

las predicciones generadas con el modelo de red, implica que estas pueden servir como una guía apropiada para el desarrollo de experimentos en trabajos futuros.

Referencias

Akutsu, T., Kuhara, S., Maruyama, O., & Miyano, S. (1998). A system for identifying genetic networks from gene expression patterns produced by gene disruptions and overexpressions. Genome Informatics, 9, 151-160.

Alon U (2006) An introduction to systems biology: design principles of biological circuits. CRC press, Boca Raton

Alon, U. (2007). Network motifs: theory and experimental approaches. Nature Reviews Genetics, 8(6), 450-461.

Arora, M. (2011). *The art of hardware architecture: Design methods and techniques for digital circuits*. Springer Science & Business Media.

Ay, F., Xu, F., & Kahveci, T. (2009). Scalable steady state analysis of Boolean biological regulatory networks. PloS one, 4(12), e7992.

Balleza, E., Alvarez-Buylla, E. R., Chaos, A., Kauffman, S., Shmulevich, I., & Aldana, M. (2008). Critical dynamics in genetic regulatory networks: examples from four kingdoms. *PLoS One*, *3*(6), e2456.

Barabasi, A. L., & Oltvai, Z. N. (2004). Network biology: understanding the cell's functional organization. *Nature reviews. Genetics*, *5*(2), 101.

Berntenis, N., & Ebeling, M. (2013). Detection of attractors of large boolean networks via exhaustive enumeration of appropriate subspaces of the state space. BMC bioinformatics, 14(1), 361.

Bierie, B., & Moses, H. L. (2010). Transforming growth factor beta (TGF-β) and inflammation in cancer. *Cytokine & growth factor reviews*, *21*(1), 49-59.

Boole G (1848) The calculus of logic. *Cambridge and Dublin Mathematical Journal*, *3*(1848), pp.183-198.

Bornholdt, S., & Rohlf, T. (2000). Topological evolution of dynamical networks: Global criticality from local dynamics. *Physical Review Letters*, *84*(26), 6114.

Bryant, R. E. (1992). Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, *24*(3), 293-318.

Chaouiya, C., Naldi, A., & Thieffry, D. (2012). Logical modelling of gene regulatory networks with GINsim. Bacterial Molecular Networks: Methods and Protocols, 463-479.

Chen, K. C., Wang, T. Y., Tseng, H. H., Huang, C. Y. F., & Kao, C. Y. (2005). A stochastic differential equation model for quantifying transcriptional regulatory network in Saccharomyces cerevisiae. *Bioinformatics*, *21*(12), 2883-2890.

Chen, T., He, H. L., & Church, G. M. (1999, January). Modeling gene expression with differential equations. In Pacific symposium on biocomputing (Vol. 4, No. 29, p. 40).

Cheng, C., Yan, K. K., Hwang, W., Qian, J., Bhardwaj, N., Rozowsky, J., ... & Snyder, M. (2011). Construction and analysis of an integrated regulatory network derived from high-throughput sequencing data. *PLoS Comput Biol*, *7*(11), e1002190.

Choo, S. M., & Cho, K. H. (2016). An efficient algorithm for identifying primary phenotype attractors of a large-scale Boolean network. BMC Systems Biology, 10(1), 95.

Chu, P. P. (2012). *Embedded SoPC Design with NIOS II Processor and Verilog Examples*. John Wiley & Sons.

Cohen, J. E. (2004). Mathematics is biology's next microscope, only better; biology is mathematics' next physics, only better. PLoS biology, 2(12), e439.

Collier J, Murray SR, Shapiro L (2006) DnaA couples DNA replication and the expression of two cell cycle master regulators. EMBO J 25(2):346–356

Davidson, E. H., Rast, J. P., Oliveri, P., Ransick, A., Calestani, C., Yuh, C. H., ... & Otim, O. (2002). A genomic regulatory network for development. science, 295(5560), 1669-1678.

De Jong, H., Ranquet, C., Ropers, D., Pinel, C., & Geiselmann, J. (2010). Experimental and computational validation of models of fluorescent and luminescent reporter genes in bacteria. BMC systems biology, 4(1), 55.

Devloo, V., Hansen, P., & Labbé, M. (2003). Identification of all steady states in large networks by logical analysis. *Bulletin of mathematical biology*, *65*(6), 1025-1051.

Di Cara, A., Garg, A., De Micheli, G., Xenarios, I., & Mendoza, L. (2007). Dynamic simulation of regulatory networks using SQUAD. *BMC bioinformatics*, *8*(1), 462.

Domian IJ, Reisenauer A, Shapiro L (1999) Feedback control of a master bacterial cell-cycle regulator. Proc Natl Acad Sci 96(12):6648–6653

Dubrova, E., & Teslenko, M. (2011). A SAT-based algorithm for finding attractors in synchronous boolean networks. *IEEE/ACM transactions on computational biology and bioinformatics*, *8*(5), 1393-1399.

Ekbom, A., Adami, H. O., Helmick, C., & Zack, M. (1990). Increased risk of large-bowel cancer in Crohn's disease with colonic involvement. *The Lancet*, *336*(8711), 357-359.

England JC, Perchuk BS, Laub MT, Gober JW (2010) Global regulation of gene expression and cell differentiation in Caulobacter crescentus in response to nutrient availability. J Bacteriol 192(3):819–833

Erwin, D. H., & Davidson, E. H. (2009). The evolution of hierarchical gene regulatory networks. Nature Reviews Genetics, 10(2), 141-148.

Galagan, J. E., Minch, K., Peterson, M., Lyubetskaya, A., Azizi, E., Sweet, L., ... & Abeel, T. (2013). The Mycobacterium tuberculosis regulatory network and hypoxia. *Nature*, *499*(7457), 178-183.

Gama-Castro, S., Salgado, H., Peralta-Gil, M., Santos-Zavaleta, A., Muniz-Rascado, L., Solano-Lira, H., ... & Porron-Sotelo, L. (2011). RegulonDB version 7.0: transcriptional

regulation of Escherichia coli K-12 integrated within genetic sensory response units (Gensor Units). Nucleic acids research, 39(suppl 1), D98-D105.

Garagnani, P., Pirazzini, C., & Franceschi, C. (2013). Colorectal cancer microenvironment: among nutrition, gut microbiota, inflammation and epigenetics. *Current pharmaceutical design*, *19*(4), 765-778.

Garg A, *et al*. (2007) An efficient method for dynamic analysis of gene regulatory networks, Springer LNBI, 2007, vol. 4453 (pg. 62-76)

Garg, A., Di Cara, A., Xenarios, I., Mendoza, L., & De Micheli, G. (2008). Synchronous versus asynchronous modeling of gene regulatory networks. Bioinformatics, 24(17), 1917-1925.

Giardiello, F. M., Offerhaus, G. J. A., & DuBois, R. N. (1995). The role of nonsteroidal anti-inflammatory drugs in colorectal cancer prevention. *European Journal of Cancer*, *31*(7), 1071-1076.

Gora KG, Tsokos CG, Chen YE, Srinivasan BS, Perchuk BS, Laub MT (2010) A cell-type-specific protein-protein interaction modulates transcriptional activity of a master regulator in Caulobacter crescentus. Mol cell 39(3):455–467

Gottesman, S. (1984). Bacterial regulation: global regulatory networks. *Annual review of genetics*, *18*(1), 415-441.

Greten, F. R., Eckmann, L., Greten, T. F., Park, J. M., Li, Z. W., Egan, L. J., ... & Karin, M. (2004). IKKβ links inflammation and tumorigenesis in a mouse model of colitis-associated cancer. *Cell*, *118*(3), 285-296.

Guo, W., Yang, G., Wu, W., He, L., & Sun, M. (2014). A parallel attractor finding algorithm based on Boolean satisfiability for genetic regulatory networks. *PloS one*, *9*(4), e94258.

He, Z., Zhan, M., Liu, S., Fang, Z., & Yao, C. (2016). An Algorithm for Finding the Singleton Attractors and Pre-Images in Strong-Inhibition Boolean Networks. PloS one, 11(11), e0166906.

Holtzendorff J, Hung D, Brende P, Reisenauer A, Viollier PH, McAdams HH, Shapiro L (2004) Oscillating global regulators control the genetic circuit driving a bacterial cell cycle. Science 304(5673):983–987

Hong, C., Hwang, J., Cho, K. H., & Shin, I. (2015). An efficient steady-state analysis method for large boolean networks with high maximum node connectivity. PloS one, 10(12), e0145734.

Hung, E., & Wilton, S. J. (2015). Zero-overhead FPGA Debugging. *Reconfigurable Logic: Architecture, Tools, and Applications*, *48*, 71.

Irons, D. J. (2006). Improving the efficiency of attractor cycle identification in Boolean networks. *Physica D: Nonlinear Phenomena*, *217*(1), 7-21.

Irurzun-Arana, I., Pastor, J. M., Trocóniz, I. F., & Gómez-Mantilla, J. D. (2017). Advanced Boolean modeling of biological networks applied to systems pharmacology. Bioinformatics, 33(7), 1040-1048.

Jensen RB (2006) Coordination between chromosome replication, segregation, and cell division in Caulobacter crescentus. J Bacteriol 188(6):2244–2253

Karlebach, G., & Shamir, R. (2008). Modelling and analysis of gene regulatory networks. Nature Reviews Molecular Cell Biology, 9(10), 770-780.

Kauffman, S. A. (1969). Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of theoretical biology*, *22*(3), 437-467.

Kraetzl, M., & Hsu, D. F. (1999). Parallel System Communications and Interconnections.

Lin, W. W., & Karin, M. (2007). A cytokine-mediated link between innate immunity, inflammation, and cancer. *Journal of Clinical Investigation*, *117*(5), 1175.

Lu, J., Zeng, H., Liang, Z., Chen, L., Zhang, L., Zhang, H., ... & Geng, M. (2015). Network modelling reveals the mechanism underlying colitis-associated colon cancer and identifies novel combinatorial anti-cancer targets. *Scientific reports*, *5*.

Martínez-Antonio, A., & Collado-Vides, J. (2003). Identifying global regulators in transcriptional regulatory networks in bacteria. Current opinion in microbiology, 6(5), 482-489.

Melkman, A. A., Tamura, T., & Akutsu, T. (2010). Determining a singleton attractor of an AND/OR Boolean network in O (1.587 n) time. *Information Processing Letters*, *110*(14-15), 565-569.

Minch, K. J., Rustad, T. R., Peterson, E. J., Winkler, J., Reiss, D. J., Ma, S., ... & Mawhinney, C. (2015). The DNA-binding network of Mycobacterium tuberculosis. *Nature communications*, *6*.

Miskov-Zivanov, N., Bresticker, A., Krishnaswamy, D., Venkatakrishnan, S., Marculescu, D., & Faeder, J. R. (2011a, August). Emulation of biological networks in reconfigurable hardware. In *Proceedings of the 2nd ACM Conference on Bioinformatics, Computational Biology and Biomedicine* (pp. 536-540). ACM.

Miskov-Zivanov, N., Bresticker, A., Krishnaswamy, D., Venkatakrishnan, S., Kashinkunti, P., Marculescu, D., & Faeder, J. R. (2011b, August). Regulatory network analysis acceleration with reconfigurable hardware. In *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE* (pp. 149-152). IEEE.

Miskov-Zivanov, N., Zuliani, P., Clarke, E. M., & Faeder, J. R. (2013, September). Studies of biological networks with statistical model checking: application to immune system cells. In *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics* (p. 728). ACM.

Mohr, A. (2014). Quantum computing in complexity theory and theory of computation. *Carbondale, IL*.

Monod, J., & Jacob, F. (1961, January). General conclusions: teleonomic mechanisms in cellular metabolism, growth, and differentiation. In *Cold Spring Harbor symposia on quantitative biology* (Vol. 26, pp. 389-401). Cold Spring Harbor Laboratory Press.

Moret, B., & Shapiro, H. D. (1991). Algorithms from P to NP, Vol. I: Design and Efficiency.

Müssel, C., Hopfensitz, M., & Kestler, H. A. (2010). BoolNet—an R package for generation, reconstruction and analysis of Boolean networks. *Bioinformatics*, *26*(10), 1378-1380.

Naldi, A., Remy, E., Thieffry, D., & Chaouiya, C. (2009, August). A Reduction of Logical Regulatory Graphs Preserving Essential Dynamical Properties. In *CMSB* (Vol. 9, pp. 266-280).

Naldi, A., Remy, E., Thieffry, D., & Chaouiya, C. (2011). Dynamically consistent reduction of logical regulatory graphs. *Theoretical Computer Science*, *412*(21), 2207-2218.

Pauling, J., Röttger, R., Tauch, A., Azevedo, V., & Baumbach, J. (2011). CoryneRegNet 6.0—Updated database content, new analysis methods and novel features focusing on community demands. *Nucleic acids research*, *40*(D1), D610-D614.

Pearl, J. (1984). Heuristics: intelligent search strategies for computer problem solving.

Purvis, J. E., & Lahav, G. (2013). Encoding and decoding cellular information through signaling dynamics. *Cell*, *152*(5), 945-956.

Quiñones-Valles, C., Sánchez-Osorio, I., & Martínez-Antonio, A. (2014). Dynamical Modeling of the Cell Cycle and Cell Fate Emergence in Caulobacter crescentus. *PloS one*, *9*(11), e111116.

Quon, K. C., Yang, B., Domian, I. J., Shapiro, L., & Marczynski, G. T. (1998). Negative control of bacterial DNA replication by a cell cycle regulatory protein that binds at the chromosome origin. *Proceedings of the National Academy of Sciences*, *95*(1), 120-125.

Reisenauer, A., & Shapiro, L. (2002). DNA methylation affects the cell cycle transcription of the CtrA global regulator in Caulobacter. *The EMBO journal*, *21*(18), 4969-4977.

Reisenauer, A., Kahng, L. S., McCollum, S., & Shapiro, L. (1999a). Bacterial DNA methylation: a cell cycle regulator?. *Journal of bacteriology*, *181*(17), 5135-5139.

Reisenauer, A., Quon, K., & Shapiro, L. (1999b). The CtrA response regulator mediates temporal control of gene expression during the Caulobacter cell cycle. *Journal of bacteriology*, *181*(8), 2430-2439.

Ryan, K. R., & Shapiro, L. (2003). Temporal and spatial regulation in prokaryotic cell cycle progression and development. *Annual review of biochemistry*, *72*(1), 367-394.

Saadatpour, A., & Albert, R. (2013). Boolean modeling of biological regulatory networks: a methodology tutorial. *Methods*, *62*(1), 3-12.

Saadatpour, A., Albert, I., & Albert, R. (2010). Attractor analysis of asynchronous Boolean models of signal transduction networks. *Journal of theoretical biology*, *266*(4), 641-656.

Saadatpour, A., Wang, R. S., Liao, A., Liu, X., Loughran, T. P., Albert, I., & Albert, R. (2011). Dynamical and structural analysis of a T cell survival network identifies novel candidate therapeutic targets for large granular lymphocyte leukemia. *PLoS computational biology*, *7*(11), e1002267.

Sakamoto, E., & Iba, H. (2001). Inferring a system of differential equations for a gene regulatory network by using genetic programming. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on* (Vol. 1, pp. 720-726). IEEE.

Sánchez-Osorio, I., Hernández-Martínez, C. A., & Martínez-Antonio, A. (2017). Modeling Asymmetric Cell Division in Caulobacter crescentus Using a Boolean Logic Approach. In *Asymmetric Cell Division in Development, Differentiation and Cancer* (pp. 1-21). Springer International Publishing.

Sanchez-Osorio, I., Ramos, F., Mayorga, P., & Dantan, E. (2014). Foundations for modeling the dynamics of gene regulatory networks: a multilevel-perspective review. *Journal of bioinformatics and computational biology*, *12*(01), 1330003.

Scott, M., Gunderson, C. W., Mateescu, E. M., Zhang, Z., & Hwa, T. (2010). Interdependence of cell growth and gene expression: origins and consequences. *Science*, *330*(6007), 1099-1102.

Sedgewick, R., & Flajolet, P. (2013). *An introduction to the analysis of algorithms*. Addison-Wesley.

So, H. K. H., & Liu, C. (2016). FPGA overlays. In *FPGAs for Software Programmers* (pp. 285-305). Springer International Publishing.

Tamura, T., & Akutsu, T. (2009). Detecting a singleton attractor in a Boolean network utilizing SAT algorithms. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, *92*(2), 493-501.

Tan, M. H., Kozdon, J. B., Shen, X., Shapiro, L., & McAdams, H. H. (2010). An essential transcription factor, SciP, enhances robustness of Caulobacter cell cycle regulation. *Proceedings of the National Academy of Sciences*, *107*(44), 18985-18990.

Thomas, S. R., & Mikulecky, D. C. (1978). A network thermodynamic model of salt and water flow across the kidney proximal tubule. *American Journal of Physiology-Renal Physiology*, *235*(6), F638-F648.

Veliz-Cuba, A. (2011). Reduction of Boolean network models. *Journal of theoretical biology*, *289*, 167-172.

Veliz-Cuba, A., Aguilar, B., Hinkelmann, F., & Laubenbacher, R. (2014). Steady state analysis of Boolean molecular network models via model reduction and computational algebra. *BMC bioinformatics*, *15*(1), 221.

Wagner, J. K., & Brun, Y. V. (2007). Out on a limb: how the Caulobacter stalk can boost the study of bacterial cell shape. *Molecular microbiology*, *64*(1), 28-33.

Wang, D., & DuBois, R. N. (2010). The role of COX-2 in intestinal inflammation and colorectal cancer. *Oncogene*, *29*(6), 781.

Winzeler, E., & Shapiro, L. (1996). A Novel Promoter Motif forCaulobacterCell Cycle-controlled DNA Replication Genes. *Journal of molecular biology*, *264*(3), 412-425.

Yaniv, M. (2011). The 50th anniversary of the publication of the operon theory in the Journal of Molecular Biology: past, present and future. *Journal of molecular biology*, *409*(1), 1-6.

Yu, H., Kortylewski, M., & Pardoll, D. (2007). Crosstalk between cancer and immune cells: role of STAT3 in the tumour microenvironment. *Nature reviews. Immunology*, *7*(1), 41.

Zhao, Y., Kim, J., & Filippone, M. (2013). Aggregation algorithm towards large-scale Boolean network analysis. *IEEE Transactions on Automatic Control*, *58*(8), 1976-1985.