Universidad Popular Autónoma del Estado de Puebla

Vicerrectoría de Investigación y Posgrados

Decanato de Ingeniería y Negocios

Departamento de Ingeniería y Tecnologías de la Información

Posgrados en Ingeniería Biomédica

ACELERANDO LA ALINEACIÓN DE LECTURAS CORTAS DE ADN MEDIANTE CÓMPUTO RECONFIGURABLE

Tesis que para obtener el Grado de
Doctor en Ingeniería Biomédica

Presenta:

M. EN C. DANIEL PACHECO BAUTISTA

Asesores:

Dr. Manuel González Pérez Dr. Juan Manuel López Oglesby Dr. Ignacio Algredo Badillo

Puebla, México Junio 2016

Universidad Popular Autónoma del Estado de Puebla

Vicerrectoría de Investigación y Posgrados

Decanato de Ingeniería y Negocios

Departamento de Ingeniería y Tecnologías de la Información

Posgrados en Ingeniería Biomédica

Doctorado en Ingeniería Biomédica

Se aprueba la Tesis:

ACELERANDO LA ALINEACIÓN DE LECTURAS CORTAS DE ADN MEDIANTE CÓMPUTO RECONFIGURABLE

Comité Asesor:

Dr. Manuel González Pérez
Director de Tesis

Dr. Juan Manuel López Oglesby
Asesor de tesis

Dr. Ignacio Algredo Badillo
Asesor de Tesis

Puebla, México

Junio 2016

Universidad Popular Autónoma del Estado de Puebla

RESUMEN

ACELERANDO LA ALINEACIÓN DE LECTURAS CORTAS DE ADN MEDIANTE CÓMPUTO RECONFIGURABLE.

Por: Daniel Pacheco Bautista

Director de tesis: Dr. Manuel González Pérez
Departamento de Posgrado

La secuenciación de ADN es el proceso de obtención del orden de cada una de las bases nitrogenadas que conforman el ADN. Ésta tiene una lista larga y versátil de aplicaciones, siendo una tecnología fundamental en la investigación de algunos tipos de cáncer, así como del VIH, ayuda también a incrementar el conocimiento básico de organismos y células, diagnóstico de pacientes, investigación a la resistencia de drogas, o predisposición a enfermedades.

En los últimos años ha ocurrido un avance impresionante en las máquinas de secuenciación paralela masiva, también llamadas de secuenciación de siguiente generación (NGS), por ejemplo, máquinas recientes como Illumina Hiseq son capaces de generar millones de lecturas en una sola corrida. No obstante, estas tecnologías están limitadas a secuenciar solo fragmentos pequeños de material genético (entre 35 y 1100 pares bases), por lo que para secuenciar un genoma completo es necesario dividir la cadena, secuenciar y posteriormente ensamblar las lecturas cortas obtenidas. Debido a la cantidad enorme de lecturas procesadas y al tamaño del genoma de referencia, este último paso se realiza con algoritmos de complejidad temporal y espacial elevada, y son ejecutados en computadoras con múltiples núcleos o procesadores trabajando en paralelo e incluso en clústeres de computadoras. Aun así, el proceso sigue siendo lento comparado con la velocidad de producción de lecturas de la tecnología NGS.

En esta tesis se investiga el estado del arte de las tecnologías de secuenciación de ADN y de los programas de alineación de lecturas cortas y se propone una solución alterna para incrementar la velocidad del proceso de reconstrucción de genomas basada en lógica reconfigurable. En específico se acelera la alineación de lecturas cortas, una forma particular de ensamble en la cual se utiliza como referencia un genoma ensamblado previamente. El algoritmo implementado en hardware es el de alineación exacta de Ferragina y Manzini, conocido en la literatura como los índices de FM, el cual es el soporte de muchos de los programas de alineación más utilizados y referenciados en la actualidad tales como Bowtie y BWA.

El diseño propuesto fue descrito completamente en Verilog, sintetizado e implementado mediante Vivado 2014.2 y probado en la tarjeta de aceleración M505k325t de la empresa Pico Computing la cual contiene el FPGA Xilinx Kintex-7 k325t. La tarjeta fue instalada en una computadora con procesador Intel Core I5 de cuarta generación con 8 GB en memoria DDR. Los resultados confirman que un solo núcleo de alineación incluido en el FPGA puede acelerar el proceso en un factor cercano a 7X comparado con la ejecución del algoritmo en software y mejora exponencialmente a medida que el número de nucleótidos en la cadena de referencia se incrementa. Además al ser un diseño completamente modular puede expandirse para incrementar el nivel de paralelismo. De lo anterior se estima que una implementación utilizando la máxima capacidad del sistema (6 módulos M505 instalados en el CPU anfitrión), puede lograr la aceleración del mismo en un factor aproximado de 50X.

Dedicada con mucho cariño a:

MIS HIJOS

Jeovani y Gael Alejandro

AGRADECIMIENTOS

Este trabajo no hubiese sido posible sin la participación de muchas personas y organizaciones a las cuales quiero agradecer, en especial a mi director de tesis el Dr. Manuel González Pérez y a mi coordinador de posgrado el Dr. Juan Manuel López Oglesby, quienes durante más de 3 años compartieron conmigo no solo su experiencia profesional sino también de vida.

A mis revisores, profesores y compañeros de generación, sus sugerencias enriquecieron notablemente la investigación.

A la Universidad del Istmo y al programa de mejora al profesorado (PROMEP) por su apoyo a través de la beca con folio unistmo-003.

A la empresa Xilinx por apoyar el proyecto a través de la donación del entorno de desarrollo Vivado 2014.2.

Y a empresas UPAEP por el financiamiento para la adquisición de las tarjetas de desarrollo.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE LISTADOS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN

1.1 El problema de la alineación de ADN

A partir de la culminación del proyecto del genoma humano en el 2003, la tecnología de secuenciación, la cual permite la obtención del orden de cada uno de los nucleótidos que conforman el ADN, ha alcanzado un avance impresionante. Las máquinas de secuenciación denominadas de siguiente generación o NGS (del inglés Next Generation Technology), tales como los analizadores genéticos ILLumina's Hiseq o Applied Biosystem 5500, son capaces de producir millones de lecturas de ADN por día y su velocidad mejora en forma exponencial (Perkel, 2014). Lamentablemente, estas máquinas están limitadas a la secuenciación de segmentos cortos de ADN, por lo que para secuenciar genomas completos es necesario primero dividirlo en un gran número de segmentos, secuenciar cada segmento y posteriormente construir la cadena que represente al genoma completo, este último paso se conoce como ensamble de ADN.

Una forma especial de lograr el ensamble es tomando como referencia otro genoma secuenciado previamente, en cuyo caso el proceso se conoce como alineación o mapeo (Figura 1.1). Lo complicado del problema es la cantidad enorme de datos que deben tratarse, por ejemplo, en el caso particular del genoma humano se tiene aproximadamente 200 millones de lecturas cortas de entre 35 y 1100 nucleótidos, cada una debe alinearse a una secuencia de referencia de 3000 millones de pares bases (Pelak, y otros, 2010). Ante esta situación las herramientas software convencionales se ven limitadas y son incapaces de competir con la elevada velocidad de las máquinas NGS. Aun grandes clústeres de computadoras equipadas con cientos de procesadores pueden

requerir días para completar los cálculos. Lo anterior ha motivado a numerosos grupos de investigación alrededor del mundo al desarrollo de formas alternas para acelerar el proceso, por una parte se ha desarrollado software con algoritmos heurísticos que si bien no son tan eficientes como aquellos basados en programación dinámica disminuyen el tiempo de procesamiento sustancialmente, BWA (Li & Durbin, 2009), Bowtie (Langmead, Trapnell, Pop, & Salzberg, 2009), GASSST (Rizk & Lavenier, 2010), SeqMap (Jiang & Wong, 2008), SOAP2 (Li, y otros, 2009) y MAQ (Li, Ruan, & Durbin, 2008) son algunos ejemplos de programas en esta categoría. Por otra parte, se ha recurrido al uso de plataformas de computo paralelo, principalmente GPUs (Grapich Processor Units) y FPGAs (Field Programmable Gate Array), los cuales han resultado eficientes en la implementación de algoritmos tradicionales utilizados en la alineación de fragmentos de ADN, tales como: Smith-Waterman (Li, Shum, & Truong, 2007), Blast (Kim, Clauson, Olson, Ebeling, Hauck, & Ruzzo, 2012) y recientemente en la implementación GPU del software SOAP3 (Liu, y otros, 2012).

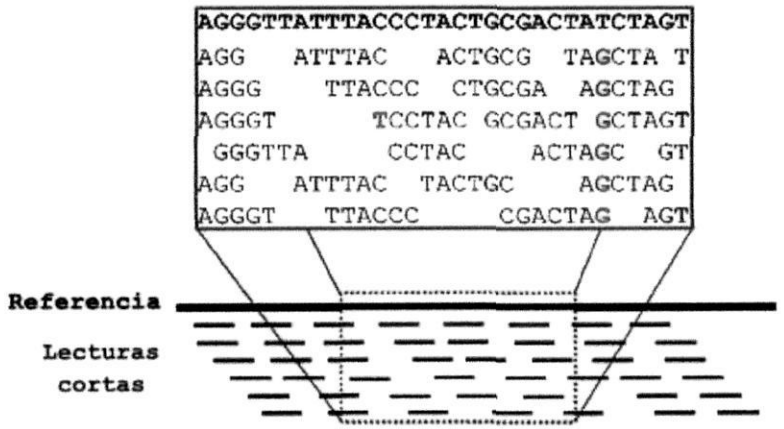


Figura 1.1 Alineación de ADN. Múltiples lecturas cortas de ADN deben ordenarse para reconstruir el genoma original.

En esta tesis se realiza la implementación hardware del algoritmo de búsquedas exactas de Ferragina y Manzini (Ferragina & Manzini, 2000) también conocido como índices de FM. Este algoritmo es el núcleo de muchos de los programas de alineación más referenciados en la actualidad, debido a su rapidez y a su relativamente bajo uso de memoria.

1.2 Motivación y objetivos

La secuenciación de ADN es un proceso clave en biología molecular, genómica y medicina (Frese, Katus, & Meder, 2013). En los últimos años, los avances tecnológicos han hecho posible secuenciar en forma más económica y más rápida cantidades enormes de material genómico abriendo oportunidades sin precedentes para la investigación y el diagnóstico.

La secuenciación de ADN tiene una lista larga y versátil de aplicaciones, siendo una tecnología clave en la investigación de algunos tipos de cáncer, así como del VIH, ayuda también a incrementar el conocimiento básico de organismos y células, diagnóstico de pacientes, investigación a la resistencia de drogas o predisposición a enfermedades. No obstante, la rapidez de los secuenciadores en la producción de lecturas cortas, ha superado por mucho a la velocidad de los programas de ensamble, lo que ha resultado en un incremento enorme en el número de genomas secuenciados y no ensamblados, por consiguiente no analizados en las bases de datos mundiales, lo anterior pone en evidencia la necesidad de alternativas para la ejecución paralela y la aceleración de los algoritmos y programas de ensamble.

Además del incremento en la cantidad de datos, una ventaja fundamental de las más recientes tecnologías de secuenciación es el decremento rápido de los costos. Como se muestra en la Figura 1.2, la secuenciación del genoma completo va directo a caer por debajo de 1000 dólares. Este precio ha sido considerado por mucho tiempo el punto clave para la generalización de la medicina personalizada. La idea es que una vez que el precio caiga por debajo de esa cantidad, finalmente sea suficientemente efectivo en costo para permitir a los médicos, entregar tratamientos basados en la genética del paciente. En lugar de administrar tratamientos basados en exámenes y síntomas, el médico será capaz de revisar el genoma del paciente para diagnosticar y perfeccionar tratamientos desde la diabetes hasta el Alzheimer.

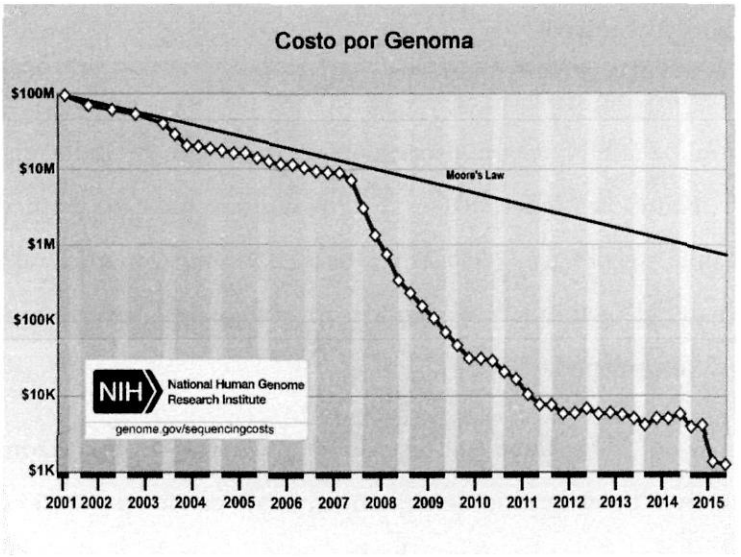


Figura 1.2 Costos de secuenciación del genoma humano por año. Fuente: NHGRI: https://www.genome.gov

En este trabajo se lleva a cabo la aceleración hardware de uno de los módulos fundamentales en muchos de los programas de alineación actuales, el diseño puede repercutir de forma importante en la disminución de los tiempos de ejecución de dichos programas y por consiguiente del proceso de secuenciación y análisis de genomas.

Objetivo General

Acelerar el proceso de alineación de lecturas cortas de ADN procedentes de las máquinas NGS mediante cómputo reconfigurable.

Objetivos Particulares

- Revisar los fundamentos y el estado del arte del proceso de secuenciación de ADN.
- Identificar los programas de alineación actuales más representativos para la alineación de secuencias de ADN, evaluar y comparar su desarrollo.
- Determinar los principales algoritmos y heurísticas que soportan a los programas de alineación.

- Analizar cada algoritmo y determinar su factibilidad para implementarse en hardware.
- Desarrollar, simular y optimizar una arquitectura que maximice el paralelismo y acelere la ejecución del algoritmo.
- Implementar y probar el diseño en la tarjeta de desarrollo M505-KT325T.

1.3 Organización de la tesis

El resto de la tesis se encuentra organizada de la siguiente manera: en el capítulo 2 se presenta el respaldo teórico de la investigación, éste abarca los principios y técnicas de secuenciación, la definición del problema de alineación de lecturas cortas de ADN, tópicos selectos de aceleración hardware, y la revisión de trabajos relacionados. El capítulo 3 contiene un estudio de los principales programas de alineación en la actualidad, se muestra una comparativa de sus características y se identifican sus principales algoritmos de alineación, mismos que son analizados minuciosamente. En el capítulo 4 se lleva a cabo el diseño del sistema de aceleración del algoritmo elegido (búsquedas exactas mediante índices de FM) y se describe el proceso de implementación en el FPGA. En el capítulo 5 se presentan los principales resultados, tanto de simulación como de ejecución real en la tarjeta de aceleración. Finalmente, en el capítulo 6 se establecen las principales conclusiones y se define el trabajo futuro.

2 FUNDAMENTO TEÓRICO

En este capítulo se presentan los fundamentos teóricos necesarios para la comprensión de la tesis. El capítulo comienza enfatizando la importancia del Ácido desoxirribonucleico, su estructura y principales funciones. A continuación se presentan los principios de la secuenciación de ADN, junto con un estudio de las plataformas NGS actuales, revisando sus características y analizando sus tres pasos fundamentales: Preparación de la muestra, inmovilización y detección. Posteriormente se define formalmente el problema de la alineación de lecturas cortas de ADN y se determinan los principales inconvenientes que dificultan la tarea. Finalmente se estudia la lógica reconfigurable como alternativa para la ejecución acelerada de tales programas y se analizan algunos trabajos relacionados.

2.1 El ácido desoxirribonucleico.

El ácido desoxirribonucleico o ADN es una macromolécula que codifica toda la información necesaria para el desarrollo, supervivencia y reproducción de un organismo. Éste se encuentra almacenado como cromosomas dentro del núcleo celular y a su totalidad se le conoce como genoma (Ver Figura 2.1). El ADN fue descubierto en 1869 por Friedrich Miescher (Dahm, 2005), sin embargo su importancia en la función celular y responsabilidad de los rasgos hereditarios fue confirmada hasta 1944 cuando Oswald Avery logró transportar ADN de una variedad de bacteria a otra observando la transferencia asociada de rasgos (Avery, MacLeod, & McCarty, 1944). Posteriormente en 1953, James Watson y Francis Crick, asistidos por los patrones de difracción de

Rayos X de Rosalind Franklin, revelaron la estructura del ADN conocida en la actualidad (Watson & Crick, 1953).

El ADN es una macromolécula muy larga de aspecto filamentoso, en forma de doble elipse y formado por un gran número de nucleótidos (Figura 2.2), cada uno compuesto por un azúcar (la desoxirribosa), una base nitrogenada y un grupo fosfato (Figura 2.3), las bases de las moléculas de ADN son las responsables de portar la información genética, en tanto que los grupos azúcar y fosfato tienen un papel fundamentalmente estructural. El azúcar de los nucleótidos es la desoxirribosa, la cual a diferencia de la Ribosa carece de un átomo de oxígeno en la posición 2 del anillo, mientras que las bases nitrogenadas son Adenina(A), Guanina(G), Citosina(C) y Timina(T), las dos primeras derivadas de la Purina, mientras que las últimas derivadas de la Pirimidina.

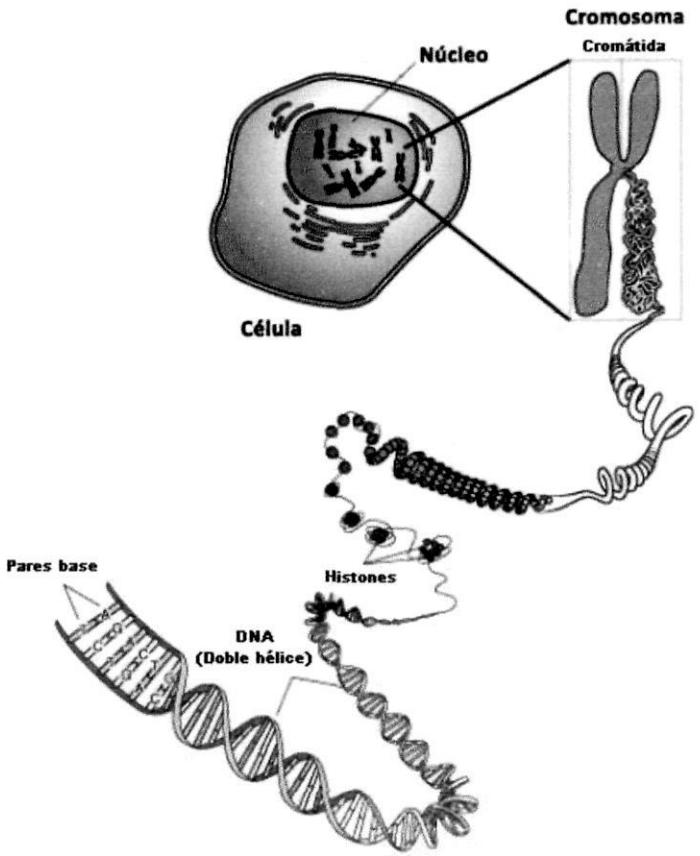


Figura 2.1 Organización de un genoma. Fuente: NHGRI Talking Glossary of Genetic Terms

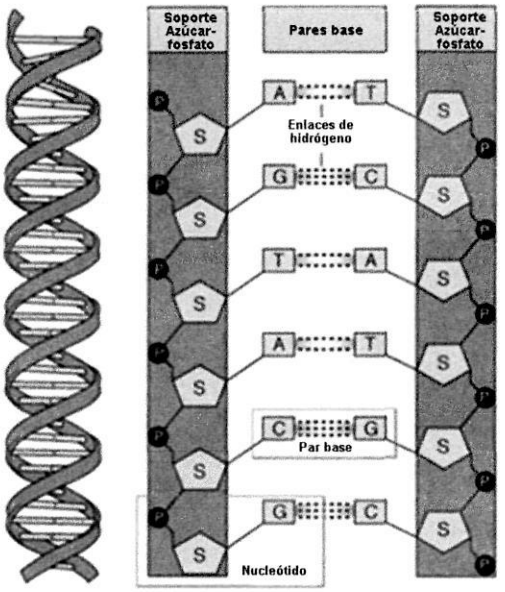


Figura 2.2 Estructura de doble elipse del ADN.

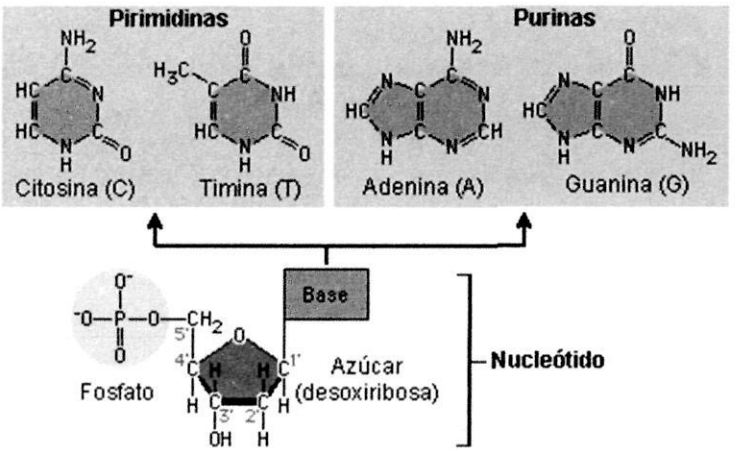


Figura 2.3 Elementos que conforman un nucleótido, las bases pueden ser: Adenina, guanina citosina o timina

El esqueleto del ADN que es constante a lo largo de la molécula, está formado por desoxirribosas ligadas por puentes fosfodiester, concretamente, el hidroxilo 3´(3'OH) del azúcar de un nucleótido está unido al hidroxilo 5'(5'OH) del azúcar adyacente por un enlace fosfodiester. La parte variable del ADN es la secuencia de los cuatro tipos de

bases, por lo que se utilizan estas iniciales para hacer referencia a una secuencia completa del mismo. Lo anterior implica que una cadena de ADN tenga siempre un grupo 5'-OH libre en uno de sus extremos y un grupo 3'OH libre en el otro extremo, es decir tienen polaridad, por convención, la secuencia de bases se escribe en la dirección 5' -> 3', de esta manera la cadena ATTG indica que el grupo 5'-OH de la desoxiadenosina está libre al igual que el grupo 3'-H de la desoxiguanosina.

Otro resultado importante confirmado por el modelo de Watson y Crick (Watson & Crick, 1953), es que las dos cadenas de la doble elipse permanecen unidas por puentes de hidrógeno entre los pares de bases y que la Adenina siempre se une a la Timina, mientras que la Citosina siempre se une a la Guanina, de esta forma conocer el orden de una de las cadenas implica la identificación de las bases de la otra cadena.

2.2 Los procesos de transcripción y traducción

El tamaño de los genomas se mide por la cantidad de nucleótidos o pares base que lo forman, variando sustancialmente de especie a especie (Tabla 2.1), una importante observación es que el tamaño del genoma está solo vagamente asociado con la complejidad del organismo. La mayoría de las unidades funcionales conocidas de los genomas son denominadas genes, un gen puede definirse como un bloque contiguo de nucleótidos operando para un propósito único, existen diversos tipos de genes siendo los más importantes los genes estructurales los cuales codifican instrucciones para crear proteínas, una segunda categoría son los genes de RNA, cuya función es determinada por su habilidad para interactuar con otras moléculas jugando un rol importante en un proceso bioquímico, el ejemplo más habitual es el gen tRNA encontrado en prácticamente cualquier forma de vida.

El número de genes estructurales va de unos cuantos en organismos no completamente vivos, hasta varios miles en algunas plantas, en particular en el genoma humano se han estimado alrededor de 30000 genes estructurales (Muse, 2005). El número y variedad de genes estructurales en organismos es en la actualidad un tema de vanguardia para los

científicos alrededor del mundo, debido a su importancia es común en la literatura usar el término *gen* para referirse a los genes estructurales, misma consideración se usará en el resto de la tesis.

Tabla 2.1 Genoma de algunas especies y número de genes identificados

Organismo	**Tamaño del genoma($x10^6$)**	**Números de genes**
HIV1	0.009	9
SARS	0.03	14
M. genitalium	0.58	470
N. equitans	0.49	552
H. helaticus	1.80	1875
E. coli	4.64	4288
P. falciparum	22.85	5268
S. cerevisiae	12.10	6000
D. melanogaster	122	13600
C. elegans	97	19049
A. thaliana	115	25000
H. sapiens	3200	30000
M. musculus	2500	37000

La forma en la cual los genes producen proteínas se conoce como el dogma central de la biología y consiste fundamentalmente en la transcripción y la traducción. La transcripción de un gen ocurre cuando una enzima conocida como Polimerasa RNA se enlaza al principio de un gen y procede a crear una molécula de RNA que iguala al ADN en el genoma. Es esta molécula denominada RNA mensajera (mRNA) que servirá como un patrón para producir una proteína. Sin embargo es necesario para los organismos regular la expresión de genes para evitar tener en producción todos los genes en todas las células todo el tiempo. Los factores de transcripción interactúan ya sea con el ADN genómico o la molécula de polimerasa para permitir un control delicado del proceso de expresión genética.

La traducción, por su parte, es el proceso de leer el patrón en una molécula de mRNA y usar la información para producir una proteína, el proceso comienza cuando una estructura conocida como ribosoma se une a la molécula mRNA, moviéndose a lo largo de la misma en unidades de 3 nucleótidos, cada uno de estos tripletes, o codones,

codifica uno de los 20 aminoácidos. La ribosoma interpreta uno a uno cada codón y agrega los aminoácidos apropiados para el crecimiento de la proteína. Las referencias (Muse, 2005) y (Guyton & Hall, 2006) muestran un tratado más profundo del tema.

2.3 Secuenciación de ADN

El proceso de obtener el código genético a partir de la macromolécula de ADN se conoce como secuenciación de ADN. La secuenciación de ADN tiene sus inicios en 1977, cuando Frederick Sanger y equipo desarrollaron el método de secuenciación enzimática, también conocida como secuenciación Sanger, didesoxi o de finalización de cadena (Sanger, Nicklen, & Coulson, 1977), y Maxam y Gilbert desarrollaron el método de secuenciación química (Maxam & Gilbert, 1977). Debido a su alta eficiencia y baja radioactividad, la secuenciación Sanger fue adoptada como la tecnología primaria en la "primer generación" de aplicaciones de secuenciación comerciales y de laboratorio.

En sus inicios, la secuenciación Sanger era laboriosa y requería de materiales radioactivos, después de años de mejora Applied Biosystems introdujo la primera máquina de secuenciación automática (nombrada AB370) en 1987, adoptando electroforesis capilar (EC), la cual hizo el proceso más rápido y más exacto. AB370 podía detectar 96 bases simultáneamente, 500 K bases al día, y las longitudes de lectura alcanzaban las 600 bases. Los instrumentos de secuenciación automática basados en EC y secuenciación Sanger, así como el software asociado, llegaron a ser las herramientas principales para la culminación del proyecto del genoma humano en el 2003 (Venter, y otros, 2001). Este proyecto estimuló fuertemente el desarrollo de nuevos instrumentos de secuenciación para incrementar la velocidad y exactitud reduciendo simultáneamente los costos y la mano de obra, surgiendo en el 2005 las tecnologías de secuenciación nombradas de siguiente generación o tecnologías NGS, las cuales difieren del método de Sanger fundamentalmente en el uso exhaustivo de tecnología paralela. En el 2005 454 Life Sciences lanza al mercado el secuenciador 454, un año después Solexa hace lo propio con el secuenciador Genome Analyzer (Que recientemente evolucionó a Hiseq), seguido por el lanzamiento de SOLID de la empresa Agencourt, CGA de complete

Genomics y PacBio RS de Pacific Biosciences, los cuales son sistemas representativos de secuenciación paralela masiva o tecnología NGS. Algunas de estas compañías fundadoras fueron compradas posteriormente por otras compañías: En el 2006 Agencourt fue comprada por Applied Biosystem, y en el 2007 454 fue comprada por Roche, mientras que Solexa fue comprada por Ilumina.

Después de años de evolución los sistemas NGS exhiben cada vez mejor desempeño y ventajas propias de la tecnología específica, como se muestra en la Tabla 2.2. En general algunas tecnologías ya superan el Terabyte de lecturas producidas por corrida, mientras que otras alcanzan exactitudes comparables con la lograda por el método de Sanger. No obstante, la longitud de lecturas limitada a valores iguales o inferiores a 1000nt sigue siendo su limitante principal. En lo que sigue se revisa los principios en la construcción de los secuenciadores Sanger y NGS, junto a algunos otros conceptos importantes en la secuenciación de ADN.

Tabla 2.2 Características de los sistemas de secuenciación NGS

Plataforma	Compañía	Longitud de lectura	Exactitud	Lecturas por corrida	Tiempo de corrida	Costo por corrida en dólares
Sanger ABI 3730XL	Applied Biosystems	400~900 nt	99.999%	-	20 min a 3hrs	--
GS FLX+	454 Life Sciences, Roche	1000 nt	99.997%	1 Gb	23 hrs.	6200
Hiseq 2500	Solexa, Illumina	125 nt	98%	1 Tb	3~10 días	20000
SOLiD 550xl	Applied Biosystems	75 nt	99.99%	300 Gb	7 días (SE) 14 días (PE)	15000
CGA Platform	Complete Genomics	62-70 nt	99.9%	--	--	--
PacBio Rs	Pacific Biosciences	860-1100	99.999%	0.01Gb	0.5-2hrs	900

2.3.1 El método de Sanger

Los principios básicos del método de Sanger de 1977 (Sanger, Nicklen, & Coulson, 1977), se utilizaron prácticamente sin cambios por tres décadas, aunque las tecnologías específicas evolucionaron sustancialmente. En este método múltiples copias de la cadena doble de ADN a secuenciar se separan en cadenas simples mediante

desnaturalización. Entonces, la muestra se divide en cuatro tubos de reacción y a cada uno de estos se le agrega un cebador o primer, un pequeño segmento de ADN monocatenario aproximadamente 20-30 nucleótidos (nt) de longitud que puede hibridar con un extremo de la hebra del ADN molde. A continuación, el cebador se extiende usando una enzima, la cual replica el ADN, conocida como Polimerasa de ADN. Cada una de las cuatros reacciones es alimentada con los cuatro tipos de nucleótidos del ADN más un tipo específico de dideoxinucleótido (ddATP, ddCTP, ddGTP, ddTTP), los cuales son nucleótidos que en su carbono 3' no contienen el grupo hidroxilo. A medida que la polimerasa de ADN recorre la cadena original, va agregando nucleótidos normales a la cadena complementaria en crecimiento, sin embargo, aleatoriamente incorporará un dideoxinucleótido deteniéndose la elongación (debido precisamente a la ausencia del grupo hidroxilo en su carbono 3').

Al terminar la reacción, cada tubo contiene duplicados incompletos de la secuencia original, todos ellos terminando en diferentes ubicaciones y en el mismo tipo de base (dependiendo del tubo al que pertenecen). Las muestras son sometidas a un proceso conocido como electroforesis en gel, el cuál separa las cadenas de acuerdo a su tamaño a lo largo de una serie de "carriles". Tradicionalmente, en secuenciación Sanger las bases de terminación de cadena se etiquetan radioactivamente, por lo que su posición puede determinarse mediante la exposición de la muestra a una película sensible a la radioactividad. La imagen resultante contiene bandas oscuras que, dependiendo de la vía, localizan las posiciones de Adenina, Guanina, Citosina y Timina en la secuencia. La Figura 2.4 muestra una vista simplificada del método.

A lo largo de los años se incorporaron ciertas mejoras al método tradicional de Sanger, las cuales incluyen, el uso de diferentes etiquetas para diferentes nucleótidos, lo que permite que el proceso tome lugar en un solo tubo de reacción. Sin embargo aún con estas mejoras el proceso es lento, tomando a lo mínimo cuatro horas para la reacción y cuatro horas adicionales para la electroforesis en gel. Debido a la probabilidad decreciente de terminación de la cadena en longitudes más largas y a las limitaciones físicas del tamaño del gel, los dispositivos de secuenciación de este tipo están limitados a la producción de lecturas de alrededor de 1000 nt.

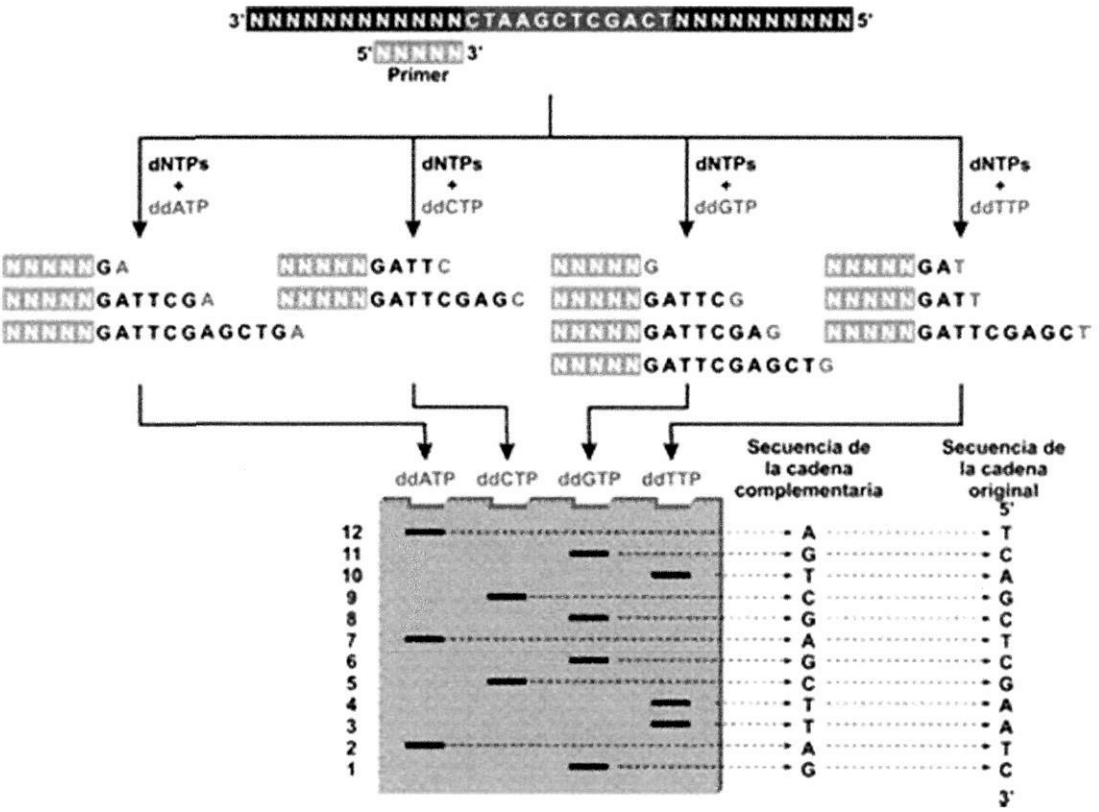


Figura 2.4 El principio de la secuenciación Sanger de ADN

2.3.2 Tecnologías NGS

Actualmente los secuenciadores NGS se comercializan por un número importante de empresas, cada una desarrollando y aplicando diferentes métodos y tecnologías (Tabla 2.3), sin embargo a pesar del dinamismo tecnológico hay principios generales utilizados en la construcción de tales dispositivos, los cuales serán revisados brevemente.

Las plataformas NGS comparten tres pasos fundamentales: Preparación de la muestra, inmovilización y detección (Figura 2.5) (Liu, y otros, 2012) (Myllykangas, Buenrostro, & Ji, 2012) (Quail, y otros, 2012). Generalmente la preparación de la muestra, involucra la adición de secuencias de ADN comunes o universales, conocidas como "adaptadores", a los extremos de hebras de ADN fragmentado aleatoriamente, la preparación resultante es referida como "librería de secuenciación". En la etapa de inmovilización, los

adaptadores se utilizan para sujetar los fragmentos de ADN a una superficie sólida y definir el sitio en el cual la reacción de secuenciación comenzará, adicionalmente, a excepción de PacBio RS, la librería de secuenciación se amplifica para formar características de secuenciación detectables y distinguidas espacialmente.

El paso final del proceso es la detección, las plataformas de secuenciación NGS integran una variedad de tecnologías ópticas y de fluidos para desarrollar y monitorear las reacciones de secuenciación molecular, las cuales pueden ser mediante síntesis de la polimerasa de ADN o ligación de oligonucleótidos fluorescentes. Cada ciclo de detección consiste en incorporar un sustrato de ácido nucleico detectable al templado inmovilizado, lavado y captura de imágenes o señales del evento molecular mediante sistemas ópticos de alta velocidad. El ciclo de incorporación, lavado y captura se repite hasta obtener la lectura de la secuencia completa de ADN.

Tabla 2.3 Plataformas de secuenciación NGS

Plataforma	**Compañía**	**Librería de secuenciación**	**Soporte**	**Generación de características**	**Reacción de secuenciación**	**Método de Detección**
GS FLX	454 Life Sciences, Roche	Adaptadores Lineales	Placa Pico-tituladora	Emulsión PCR	Síntesis	Piro-Secuenciación
Hiseq 2000	Solexa, Illumina	Adaptadores Lineales	Celdas de **flujo**	Puente PCR	Síntesis	Nucleótidos terminadores reversibles etiquetados con fluorescencia
SOLiD V4	Applied Biosystems	Adaptadores Lineales	Celdas de **flujo**	Emulsión PCR	Ligación	Sondas de oligo-neucleótidos etiquetados con fluorescencia
CGA Platform	Complete Genomics	Adaptadores Circulares	Arreglos de nano-esferas de ADN	Amplificación circular rodante	Ligación	Sondas de oligo-neucleótidos etiquetados con fluorescencia
PacBio RS	Pacific Biosciences	Adaptadores de burbujas	Guías de onda en modo cero	Molécula única	Síntesis en tiempo real	Nucleótidos etiquetados con fluorescencia fosfovinculados

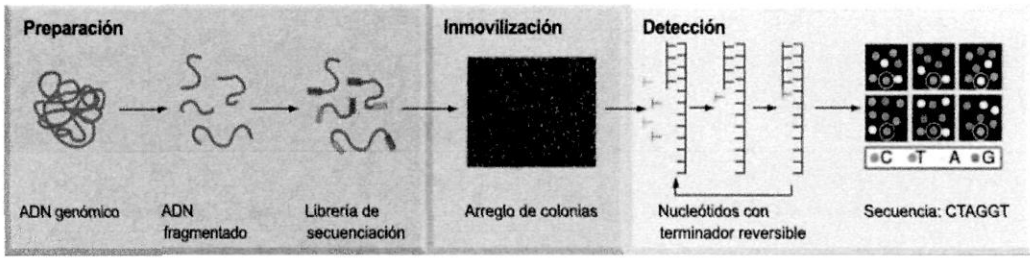


Figura 2.5 Flujo de trabajo de las tecnologías NGS: Preparación de las muestras, inmovilización y detección. Fuente: http://www.medscape.com/viewarticle/743869_4

Existen diversos métodos de amplificación, GS FLX y SOLiD utilizan emulsión PCR (Margulies, y otros, 2005) (Myllykangas, Buenrostro, & Ji, 2012), en esta técnica, pequeñas esferas con enriquecimiento individual y fragmentos de la librería de secuenciación se someten a emulsión dentro de una burbuja de reacción acuosa, posteriormente se aplica PCR para poblar la superficie de la esfera con copias clónales del molde, las esferas con la colección de ADN clonado e inmovilizado se depositan en una placa "picotituladora" (GS FLX) o en una superficie de vidrio (SOLiD), listas para el paso de detección. Hiseq por su parte utiliza emulsión en puente (Bentley, y otros, 2008), para generar clústeres *in situ* de fragmentos de la librería de secuenciación amplificada sobre un soporte sólido denominado "celdas de flujo". Por su parte, la tecnología CGA utiliza amplificación de círculo rodante (Drmanac, y otros, 2010), generando largos segmentos de ADN que se pliegan en pequeñas esferas del orden de 200 nm, denominadas DNBs. El pequeño tamaño y características bioquímicas de los DNBs permite empaquetarlos en forma muy compacta sobre un chip de silicio con un patrón matricial de pequeños puntos. El chip de silicio lleno con DNBs es llamado arreglo de nanoesferas de ADN y contiene alrededor de 180 billones de bases de ADN preparado para la detección.

El paso de detección también ofrece algunas variantes y son estas las que generalmente se utilizan para distinguir a cada fabricante. Las opciones se muestran en la figura 2.6.

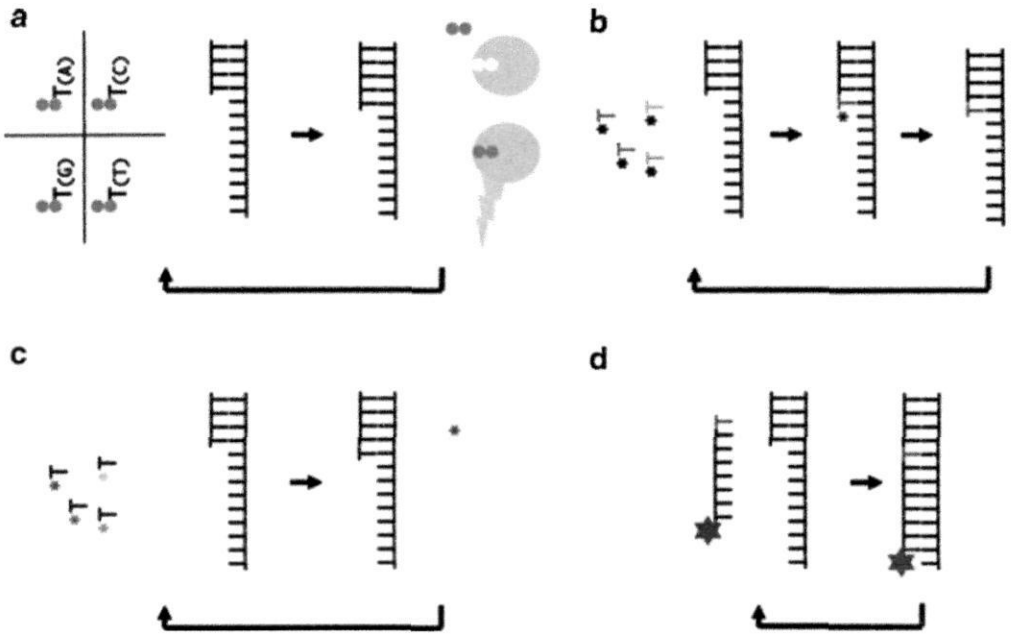


Figura 2.6 Reacciones de secuenciación cíclica. a) Piro-secuenciación, b) Nucleótidos terminadores reversibles, c) SMRT, d) Ligación.

La plataforma GS FLX utiliza la técnica llamada pirosecuenciación (Fakruddin, Mohammad Mazumdar, Chowdhury, Hossain, Mahajan, & Islam, 2013) (Margulies, y otros, 2005), mostrada en la Figura 2.6a, en cada ciclo de detección se ofrece un solo tipo de base (i.e. dATP, dCTP, dGTP o dTTP) a cada pocillo de la placa picotituladora. En caso de que esa base complemente a la base de la cadena en crecimiento, la actividad de la polimerasa de ADN y la incorporación del nucleótido producirá la emisión de un pirofosfato, enzimas luciferasa y sulfurilasa ATP convierten el pirofosfato en una ráfaga de luz visible, la cual es detectada por un sistema de imágenes CCD. Después de obtener las imágenes los excedentes (nucleótidos que no fueron incorporados a ninguna cadena de ADN en crecimiento) se retiran, para poder, en el siguiente ciclo ofrecer un nuevo tipo de base. El problema principal de esta técnica es que al utilizar química de extensión asíncrona, no hay manera de evitar la adición de múltiples bases en un único ciclo, por lo que, en caso de homopolímeros[1] la exactitud de la secuenciación sigue siendo un reto.

[1]Segmento de ADN con el mismo nucleótido, por ejemplo GGGGGGG.

El sistema Hiseq utiliza nucleótidos terminadores de cadena reversibles (Bentley, y otros, 2008), en su etapa de detección (Figura 2.6b), durante la reacción a diferencia de la pirosecuenciación, se ofrecen los cuatro nucleótidos terminadores marcados cada uno con un fluorocromo diferente, similar al método de Sanger, luego de un lavado se obtienen las imágenes y se obtiene la primera base de cada cluster. Posteriormente se elimina el fluorocromo y se desbloquea el carbono 3, permitiendo que un nuevo nucleótido pueda extender la cadena de ADN naciente. En el siguiente ciclo, nuevamente los cuatro nucleótidos terminadores marcados se ofrecen a los clústeres, procediendo de esta forma hasta terminar la cadena. La extensión síncrona de la cadena secuenciada por un nucleótido por ciclo, asegura que los homopolimeros sean secuenciados con exactitud.

PacBio RS utiliza el método de secuenciación SMRT (del inglés single molecule real time sequencing) (Eid, y otros, 2009), que permiten la secuenciación de una única molécula de ADN (i.e. no requiere el proceso de amplificación de las otras tecnologías revisadas). El secuenciador PacBio contiene una ADN polimerasa fijada en el fondo de los pocillos que van a contener las muestras. Se utilizan nucleótidos con bases nitrogenadas modificadas que al incorporarse a la cadena en proceso de polimerización liberan fluorescencia, distinta para cada tipo de base, evitando de esta manera el lavado de nucleótidos como en el caso de la pirosecuenciación. La marca fluorescente se encuentra en el grupo fosfato de la base, de ahí que la señal se observe cuando se produce la incorporación de ésta a la cadena incipiente. El proceso de secuenciación es más rápido debido a que no es necesario lavar nucleótidos ni enzimas. Las ventajas del secuenciador PacBio es que es capaz de realizar lecturas de una longitud media de 860-1100 nt, con lecturas máximas de 3000 nt. La precisión y la sensibilidad son extremadamente altas, al evitar la amplificación del ADN.

La secuenciación por ligación (Figura 2.6d) (Drmanac, y otros, 2010), es utilizada en los sistemas Solid y CGA, aunque tienen diferentes métodos el principio general es el mismo. La secuenciación por Ligación es otro método enzimático de secuenciación que emplea una ADN ligasa en lugar de una polimerasa para identificar la secuencia objetivo. Este método utiliza un reservorio de todos los oligonucleótidos posibles de

una longitud dada, marcados de acuerdo con la posición secuenciada. Los oligonucleótidos se templan y ligan; el ligamiento preferente de las ADN ligasas por su secuencia específica produce una señal correspondiente a la secuencia complementaria en esa posición concreta.

2.3.3 Tecnologías emergentes

Las tecnologías NGS creadas a partir del 2011 han sido nombradas por algunos autores como la tercera generación o tecnologías "*next-next*" (Hui, 2012) (Quail, y otros, 2012), las cuales prometen continuar en la mejora de la velocidad del proceso, incremento en la longitud de las lecturas y disminución de la razón de error en las mismas.

Además de lo anterior se observa una fuerte tendencia hacia el desarrollo de secuenciadores de bajo costo diseñados para pequeños proyectos. Life Technology e Ion Torrent han desarrollado la "Ion Personal Genome Machine (IPGM)" (www.lifetechnologies.com), el sistema IPGM contiene un arreglo de chips semiconductores capaz de sensar cambios mínimos en pH y detectar eventos de incorporación de nucleótidos por la emisión de un ion de hidrogeno desde nucleótidos naturales. La tecnología de IPGM no requiere ninguna enzima especial o nucleótidos etiquetados y toma ventaja de los avances hechos en la tecnología semiconductora.

Otra tecnología prometedora en el campo es la de Nanoporos, liderada por Oxford Nanopores (www.nanoporetech.com), la cual pretende utilizar la modulación de corriente generada con el paso del ácido nucleico a través de un poro. La secuenciación por Nanoporos promete ser la solución a las limitaciones de los secuenciadores NGS en relación a la longitud de las lecturas, permitiendo la secuenciación de cadenas de ADN enormes sin tener que modificar o preparar muestras específicas.

2.3.4 Secuenciación de cadenas largas de ADN

Las lecturas obtenidas de los instrumentos de secuenciación de cualquier generación son demasiados cortas como para cubrir regiones de interés en investigación genómica, por

lo que fue necesario el desarrollo de métodos que permitieran secuenciar segmentos más largos de ADN e incluso de genomas completos (WGS, del inglés W*hole Genome Sequencing*). La estrategia más utilizada se conoce con el nombre de secuenciación Shotgun (Pop, 2004), y se ilustra de manera simplificada en la Figura 2.7. En esta técnica la cadena de ADN a secuenciar se clona a través del uso de PCR o mediante una bacteria anfitrión, el número de copias que se obtienen se conoce como *cobertura.* Posteriormente la muestra resultante se divide aleatoriamente en pequeños fragmentos y se secuencía en forma desordenada mediante alguna de las tecnologías revisadas previamente, la división aleatoria crea fragmentos de diferente tamaño por lo que en este paso es necesario elegir únicamente las que se encuentran en un rango apropiado para la tecnología de secuenciación a utilizar. Una vez obtenidas las lecturas, los traslapes entre estas se utilizan para reconstruir mediante técnicas computacionales sofisticadas la secuencia original, a este último paso se le conoce como ensamble de secuencias y es el tema principal de este trabajo.

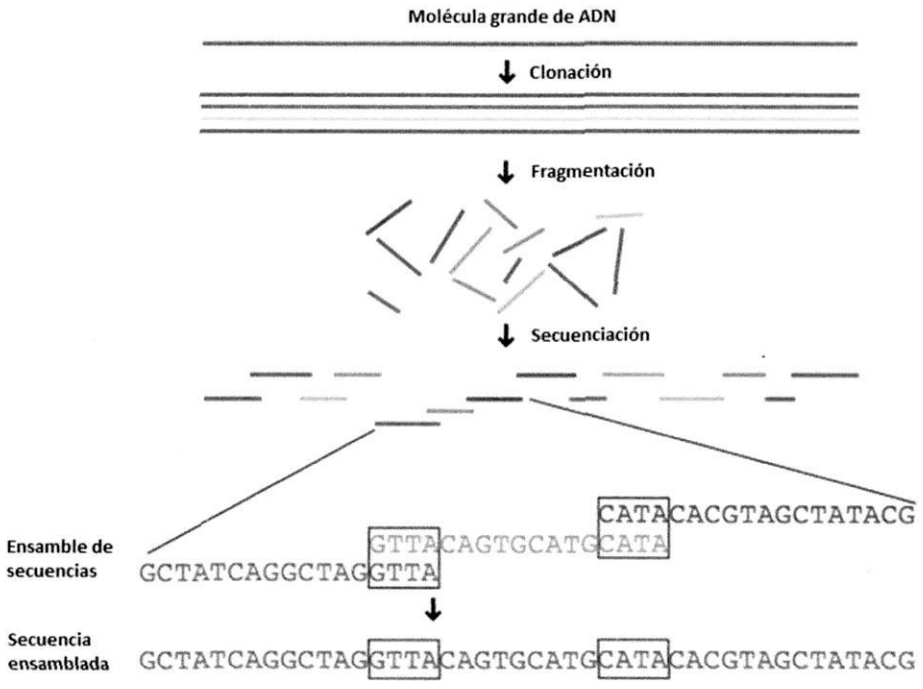


Figura 2.7 Proceso simplificado de la estrategia de secuenciación Shotgun

2.3.5 Lecturas en pares

Algunas plataformas NGS ofrecen protocolos de secuenciación por pares (PE, del inglés Pared-End), esas lecturas apareadas se obtienen al secuenciar cada fragmento de ADN desde ambos extremos (Figura 2.8a), opuesto a lo que ocurre en la secuenciación convencional (SE, del inglés Single-end), donde solo se realiza a partir de solo uno de los extremos. El objetivo de la técnica es superar las dificultades encontradas en el proceso de alineación, posterior a la secuenciación, creada principalmente por regiones repetidas dentro del genoma. Aun cuando las lecturas apareadas son típicamente mucho más cortas que los fragmentos clonados, por lo que no se traslapan, la distancia entre lecturas apareadas es conocida y esta información es utilizada por algunos algoritmos de ensamble (Figura 2.8b).

La selección de la longitud óptima en las librerías de secuenciación PE depende de varios factores que incluyen la estructura repetitiva del genoma secuenciado y limitaciones prácticas de las técnicas de secuenciación específica. Generalmente los proyectos de secuenciación crean librerías con diferentes longitudes de inserción para obtener mejores resultados que los obtenidos con una longitud única.

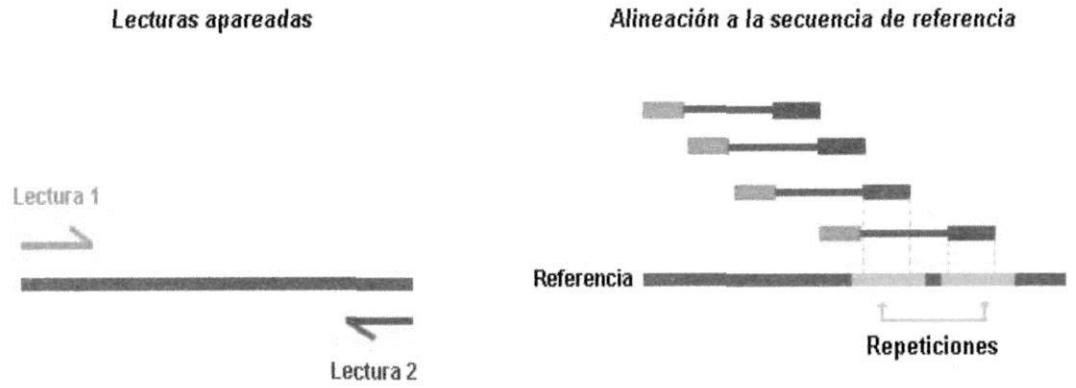


Figura 2.8 Lecturas apareadas (PE). a) Obtención de las lecturas.
b) alineación a la secuencia de referencia

2.3.6 Salida de las máquinas NGS

Después del proceso de secuenciación las máquinas NGS presentan las lecturas generalmente en el formato FASTQ (Cock, Fields, Goto, Heuer, & Rice, 2010). FASTQ es un formato muy simple basado en texto, que almacena tanto la secuencia biológica como su correspondiente puntuación de calidad mediante caracteres ASCII. Fue desarrollado originalmente en el instituto Wellcome Trust Sanger como una mejora al formato FASTA y a la fecha es el estándar de facto para almacenar la salida de instrumentos de secuenciación de alta velocidad. El Listado 2.1 muestra la estructura básica de un archivo FASTQ.

Listado 2.1 Estructura básica de un archivo FASTQ.

```
Línea 1: @Titulo y descripción opcional
Línea 2: Línea de secuencia
Línea 3: +Repetición opcional de la línea de titulo
Línea 4: Línea de calidad
```

La línea uno inicia siempre con el carácter @, y puede contener información tal como número de corrida, algún identificador de secuencia, longitud de la lectura u otro tipo de información. La línea 2 es la cadena de caracteres (A, G, C, T) que representa la secuencia de nucleótidos procesada. La línea 3 inicia siempre con el carácter + seguido, en forma opcional, de la repetición de la línea de título. Finalmente la línea cuatro incluye información de la calidad con la que cada nucleótido fue secuenciado. La puntuación de calidad se representa mediante un carácter ASCII, y es una medida de la certeza de que el nucleótido codificado sea el verdadero en el segmento de ADN original. Estas cuatro líneas se repiten a lo largo del archivo para cada secuencia generada. En el caso de lecturas en pares, habrá dos archivos FASTQ uno para cada lectura, por ejemplo la cuarta lectura en el primer archivo esta apareada con la cuarta lectura del segundo archivo.

El formato FASTQ original, también conocido como formato FASTQ-Sanger, utiliza el sistema de puntuaciones de calidad PHRED. En PHRED la puntuación de calidad de la

codificación de una base se relaciona con probabilidad de error de secuenciación (Pe) mediante la ecuación:

$$Q_{PHRED} = -10xlog_{10}(Pe) \quad [1]$$

FASTQ-Sanger, almacena puntuaciones PHRED como caracteres únicos, dando una codificación simple pero eficiente en espacio. Con el fin de que el archivo sea leíble y editado fácilmente por humanos FASTQ-Sanger utiliza los caracteres ASCII del 33 al 126, para codificar cantidades PHRED de 0 a 93 (i.e. puntuaciones PHRED con un desplazamiento ASCII de 33), cubriendo un rango de probabilidades de error desde 1.0 a $10^{-9.3}$.

La tabla 4 muestra algunas variantes del formato FASTQ-Sanger, las principales diferencias tienen que ver con el modo de expresar la puntuación de calidad. En FASTQ-Solexa el rango de caracteres ASCII utilizados va del 59 al 126, con un desplazamiento de 64, además deja de utilizar el sistema de puntuaciones PHRED, creando su propia relación entre la puntuación de calidad y la probabilidad de error, expresada como:

$$Q_{Solexa} = -10xlog_{10}\left(\frac{Pe}{1-Pe}\right) \quad [2]$$

La segunda variante es la FASTQ-illumina, la cual utiliza los caracteres ASCII del 64 al 26, un offset de 64 y el sistema PHRED al igual que el formato FASTQ-Sanger.

Tabla 2.4. Tres variantes del formato FASTQ

Formato	Caracteres ASCII		Puntuación de calidad	
	Rango	Desplazamiento	Tipo	Rango
Fastq-Sanger	33-126	33	PHRED	0 a 93
Fastq-Solexa	59-126	64	Solexa	-5 a 62
Fastq-illumina	64-126	64	PHRED	0 a 62

El Listado 2.2 muestra las primeras líneas de un archivo FASTQ-Sanger, en la línea 1 los caracteres 0/1 indican que los datos corresponden a la lectura cero y elemento uno del

par de lecturas apareadas, es decir en este archivo ese segundo número siempre ha de ser 1, y en el archivo de la lectura complementaria siempre será igual a 2. En la línea dos aparece la cadena de nucleótidos que en este caso tiene longitud de 35. La línea tres solo contiene el carácter +, aunque opcionalmente pudo haberse repetido el título de la lectura. La línea cuatro contiene la puntuación de calidad, por ejemplo, el carácter "<" con código ASCII de 60 representa una puntuación de 27 (60-33), lo que indica una probabilidad de error en la secuenciación del correspondiente nucleótido de 0.001995 ($10^{-27/10}$). Finalmente, las líneas cinco a ocho contienen la información de la segunda lectura.

```
Línea 1:  @0/1
Línea 2:  CCAGGTTGAACAACTTTTGGGATCGTGCAAGCTAA
Línea 3:  +
Línea 4:  8 ; : ; <<<<<<<< 9 ;<<<<<<<<<<<<<<<<<<<<<<<<
Línea 5:  @1/1
Línea 6:  ACCGTTAGATCTACAATTCGGTAAGTAAAAGCTTG
Línea 7:  +
Línea 8:  <<<<<<<<<88<<<<<<<<<<<< ; ; : ; <<<<<<<< 9 ;<
```

Listado 2.2 Primeras líneas de un archivo FASTQ-Sanger.

2.4 Alineación de lecturas cortas

El último paso de la secuenciación de genomas completos mediante la técnica Shotgun (ver sección 2.1.1) es el ensamble de fragmentos, el cual se realiza mediante técnicas computacionales de elevadas prestaciones. El objetivo es reconstruir a partir de los millones de lecturas obtenidas mediante las maquinas NGS, la cadena que represente el código genético de la molécula original. Lo anterior puede llevarse a cabo mediante dos formas, en la primera la reconstrucción se realiza utilizando como referencia un genoma secuenciado previamente, tal mecanismo recibe el nombre de alineación o mapeo. La segunda reconstruye la cadena a partir exclusivamente de las lecturas secuenciadas y es conocida como ensamble De Novo o simplemente Ensamble. El resto de esta tesis está dedicada a la alineación o mapeo, referimos al lector interesado en ensamble De Novo a

consultar las referencias (Li, y otros, 2012) y (Miller, Koren, & Sutton, 2010) para un estudio más profundo.

La alineación de lecturas cortas es utilizada en proyectos de re-secuenciación, donde se obtiene el código genético de miembros de una especie que ha sido previamente secuenciado mediante un método De Novo. El ejemplo más clásico es el del ser humano, en tales proyectos se persiguen encontrar variaciones genómicas que caractericen en forma particular a un individuo mediante comparación con los resultados obtenidos del proyecto del genoma humano, intentando explicar tópicos como susceptibilidad a enfermedades, rasgos fisiológicos característicos, resistencia a drogas, etc.

En tales aplicaciones, la cantidad elevada de datos generados por las tecnologías NGS así como la longitud limitada de las lecturas producidas evita el uso de programas de alineación tradicionales como BLAST (Altschul, Gish, Miller, Myers, & Lipman, 1990). Adicionalmente, a diferencia de las aplicaciones de alineación típicas, el interés ahora esta centrado en localizar variaciones mínimas entres las lecturas y la referencia, es decir se procesan genomas que se espera tengan similitud elevada con el genoma de referencia. Para dimensionar el problema, en el ejemplo del genoma humano, el número de lecturas m es usualmente 10^7-10^8, la longitud de una lectura l es de 35-1100 nucleótidos, la longitud del genoma $|R|$ es $3x10^9$ nucleótidos y las variaciones entre un humano y otro son apenas alrededor del 0.1% (Muse, 2005). Lo anterior ha incubado el desarrollo de nuevos programas de alineación de baja sensibilidad y alta velocidad denominados alineadores o mapeadores de lecturas cortas, cuyos principios de funcionamiento se tratarán en el siguiente capítulo.

2.4.1 Definición del problema de la alineación

A partir de la discusión previa y de la teoría de secuenciación NGS, pueden determinarse algunas características importantes del problema del mapeo:

1. El alfabeto está constituido por 4 letras $\sum = \{A, C, G, T\}$ cada una representando a un nucleótido, sin embargo cuando no se sabe el tipo de base en una secuencia es común usar el símbolo N en su lugar.

2. El genoma de referencia es fijo y conocido previamente, su tamaño es del orden de unas decenas a miles de millones de nucleótidos (ver sección 1.2.3). Lo que sugiere que pueda indexarse una sola vez y reutilizarse en cada proyecto de re-secuenciación de esa misma especie.

3. Las lecturas tienen una longitud fija entre 35-1100 nucleótidos para las tecnologías de secuenciación actuales. La cantidad de lecturas suele ser muy grande del orden de millones por cada pasada de las máquinas NGS.

4. La similitud entre el genoma de referencia y el genoma re-secuenciado es aproximadamente del 99.9 %

Ante este panorama el problema del mapeo puede establecerse en forma general de la siguiente manera:

Entradas: - Un conjunto de lecturas cortas de ADN $f_1,\ldots,f_m$, cada una de longitud l
- Una secuencia de referencia R de tamaño $|R|$
- Un grupo de restricciones del proceso

Salida: - Posiciones en R en donde cada lectura se mapea exacta o aproximadamente.

El grupo de restricciones puede variar dependiendo de la plataforma específica de secuenciación NGS utilizada, así como del tipo de datos procesados y de la configuración impuesta por el usuario. Es importante notar como la definición establecida permite alineaciones aproximadas, debido fundamentalmente a ciertas particularidades que ocurren en el genoma y en la secuenciación, las cuales serán revisadas con más detalle en el siguiente apartado.

2.4.2 Alineaciones aproximadas

En esencia los principales tópicos que hacen necesaria la alineación aproximada en el mapeo son: diferencias biológicas, errores de secuenciación y repeticiones de bases, a continuación se describen cada uno de estos.

Diferencias biológicas: Son diferencias locales pequeñas del genoma siendo considerado con respecto al genoma de referencia conocido para su especie, ocurren con una frecuencia de aproximadamente 1/1000 en el caso del genoma humano (Muse, 2005). Un ejemplo de estos son los polimorfismos de nucleótido simple (SNP), los cuales básicamente representan la sustitución de un solo nucleótido de la cadena de referencia (Figura 1.1a), los SNPs ocurren principalmente por la diversidad encontrada dentro de una misma especie, por ejemplo en el caso de los humanos un SNP podría ser responsable del color del cabello o la susceptibilidad a una enfermedad en particular. Otras variaciones biológicas que ocurren en secuencias genómicas son las inserciones y las supresiones (*indels*), en la cual un segmento de la lectura difiere de la referencia debido a la inserción o supresión de una o más bases. Los *indels* causan un desplazamiento de las bases con respecto al genoma de referencia, tal como se muestra en los incisos b y c de la Figura 2.9.

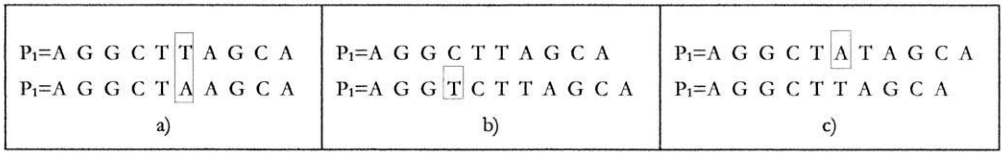


Figura 2.9 Tres casos de diferencias biológicas c) polimorfismo de nucleótido simple (SNP) b) inserción de base, b) eliminación de base.

Errores de secuenciación: Un error de secuenciación ocurre cuando la máquina de secuenciación etiqueta incorrectamente un nucleótido. Por ejemplo, en la Figura 2.10 se reporta un nucleótido de guanina (G) en el genoma, en donde un nucleótido de citosina (C) ocurre realmente. Es posible diferenciarlo de un SNP debido a que un error máquina generalmente ocurre en una única fila de las lecturas, mientras que un SNP

ocurre en todas las lecturas que cubren esa región. De esta forma la cobertura o profundidad de las lecturas cortas puede utilizarse para detectar y corregir el error. La frecuencia de ocurrencia de los errores de secuenciación es muy baja, alrededor del 0.1% en las lecturas de la mayoría de las tecnologías NGS (Liu, y otros, 2012).

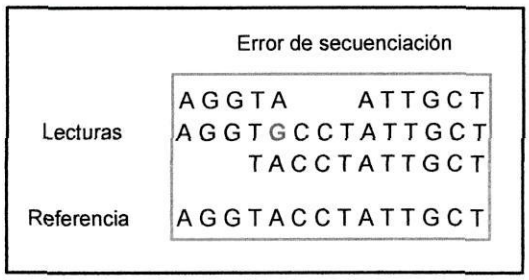


Figura 2.10 Error de secuenciación. La lectura de la segunda fila fue etiquetada como una G en lugar de una C, en estos casos la cobertura puede ayudar a reconocer y a corregir el error.

Repeticiones: Normalmente los genomas contienen gran cantidad de regiones repetidas, las cuales crean problemas de alineación fundamentalmente por las limitadas longitudes de las lecturas generadas por las máquinas NGS, en esencia entre más cortas sean tales lecturas, mayor es la probabilidad de que se concuerden erróneamente en localidades del genoma que se repiten. Este punto se trata en varias formas, dependiendo del programa usado, muchos programas utilizan la primer concordancia para la alineación, mientras que otros descartan áreas repetitivas completamente, lamentablemente ninguno de los métodos resuelve el problema, recurriendo a soluciones más sofisticadas como el uso de lecturas por pares discutidas en la sección 2.3.5.

2.5 Aceleración Hardware

Los datos NGS pueden consistir de miles de millones de lecturas cortas, por lo que la mayoría de los programas de alineación, tienen soporte para ejecución paralela en sistemas con memoria distribuida (clústeres compuestos por múltiples computadoras)

y/o usando memoria compartida (equipos de cómputo con múltiples núcleos). Aun así, los tiempos para terminar la tarea, siguen siendo muy elevados (Fonseca, Rung, Brazma, & Marioni, 2012) (Schbath, Martin, Zytnicki, Fayolle, Loux, & Gibrat, 2011), abriendo una nueva área de aplicación a la aceleración hardware.

2.5.1 Alternativas de implementación

En cómputo, la aceleración hardware es el uso de circuitos especializados para desarrollar alguna función más rápida que la lograda mediante la ejecución del algoritmo sobre un procesador de propósito general. En la actualidad las alternativas más generalizadas para implementarla son: los circuitos integrados de aplicación específica (ASIC), las unidades de aceleración gráfica (GPUs), y los dispositivos lógicos programables en el campo (FPGAs), todas estas pueden permitir que miles de unidades trabajen simultáneamente logrando el procesamiento paralelo masivo de los datos. Esta técnica aunque pueda parecer universalmente efectiva, solo toma ventaja cuando las funciones incluyen cálculos repetitivos e independientes, tal es el caso del proceso de alineación en donde millones de lecturas se alinean al genoma de referencia de forma independiente unas de otras.

Los ASIC tienen un desempeño superior al de sus contrincantes, permitiendo el mínimo consumo de potencia y la más alta velocidad de procesamiento, sin embargo a diferencia de los GPUs y los FPGAs no pueden reprogramarse, este factor limitante resulta en un incremento en el tiempo de desarrollo y en los costos de ingeniería no recurrente (NRE). Por su parte, las GPUs son coprocesadores paralelos muy económicos con una enorme penetración en el mercado, dedicados al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos o aplicaciones 3D interactivas, tienen una memoria con gran ancho de banda y un número elevado de núcleos programables, con miles de hilos hardware ejecutando programas en un modo Programa Único Múltiples Datos (SPMD), son flexibles y fácil de programar usando lenguajes de alto nivel y APIs las cuales abstraen la mayoría de los detalles del Hardware.

Comparado a la arquitectura hardware fija del GPU, los FPGAs son esencialmente arreglos densos de bloques lógicos programables, prefabricados y que el desarrollador puede (re)configurar módulo a módulo para darle la funcionalidad deseada. De esta forma se tiene el control de los compromisos entre recursos, desarrollo y nivel de paralelismo, obteniendo diseños superiores en desempeño a los basados en GPUs y cercanos en eficiencia a un ASIC (Che, Li, Sheaffer, Skadron, & Lach, 2008). Lamentablemente aunque algunos vendedores proveen núcleos con propiedad intelectual (IP) que ofrecen la mayoría de las funciones de procesamiento más comunes, la configuración de un FPGA requiere el uso de lenguajes de descripción de hardware los cuales son más complejos de utilizar que los lenguajes comunes de alto nivel..

Como puede notarse, cada una de las opciones puede mostrarse ventajosa con respecto a las otras dependiendo de la función final deseada. En esta tesis debido a las altas exigencias del diseño y al corto tiempo de desarrollo requerido (al ser un área que evoluciona aceleradamente) se ha elegido a los FPGAs, mismos que serán tratados en el resto de la sección.

2.5.2 Arreglo de compuertas programables en el campo

Un FPGA es un dispositivo cuyo núcleo es una matriz de elementos lógicos programables, cuando se programa la circuitería interna se conecta de tal modo que crea una implementación hardware de la aplicación software. Diferente a los procesadores, los FPGAs usan hardware dedicado para el procesamiento lógico y no tienen un sistema operativo.

2.5.2.1 Organización interna

Internamente un FPGA consiste de una matriz de miles de bloques lógicos configurables (CLBs), empotrados en una red de interconexiones y rodeada por una periferia de bloques de entrada y salida (IOBs), como se muestra en la Figura 2.11. Los

elementos lógicos y de ruteo se controlan por puntos de programación, los cuales pueden basarse en tecnologías de anti-fusibles, Flash o SRAM. Para cómputo reconfigurable los FPGAs basados en SRAM son los preferidos y en realidad son el estilo primario dentro de la industria FPGA en general (Teubner & Woods, 2013).

Cada **Bloque lógico configurable** está formado por dos o más unidades lógicas elementales (Figura 2.12a). La estructura exacta de una unidad lógica elemental varía entre diferentes vendedores y aun entre diferentes generaciones de FPGAs del mismo vendedor, no obstante pueden identificarse en esta cuatro elementos estructurales principales: (i) Tablas de búsqueda (LUTs) (típicamente entre 2 y 8), (ii) un número proporcional de registros de un bit, (iii) lógica aritmética y de acarreo, y (iv) varios multiplexores. La Figura 2.12b, muestra la estructura clásica de una unidad lógica elemental en la familia Vitex 4 de Xilinx y antecesores, esta cuenta con 2 LUTS, 2 registros de un bit, lógica de acarreo y 2 multiplexores, FPGAs más modernos tienen típicamente más recursos.

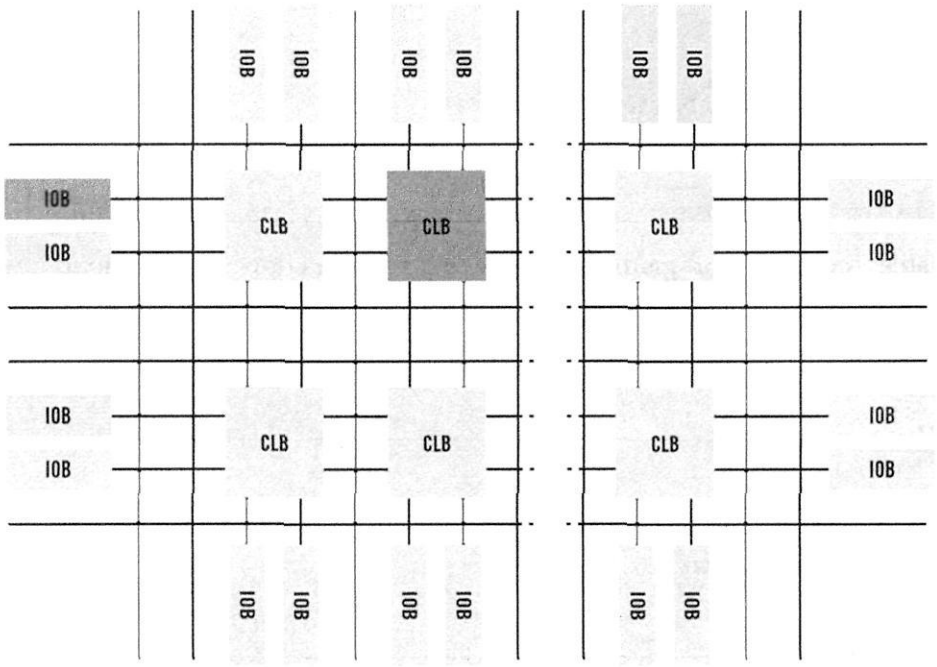


Figura 2.11 Estructura interna de un FPGA. Esta consiste de bloques lógicos configurables (CLBs), bloques de entrada y de salida (IOBs) y la estructura de interconexión

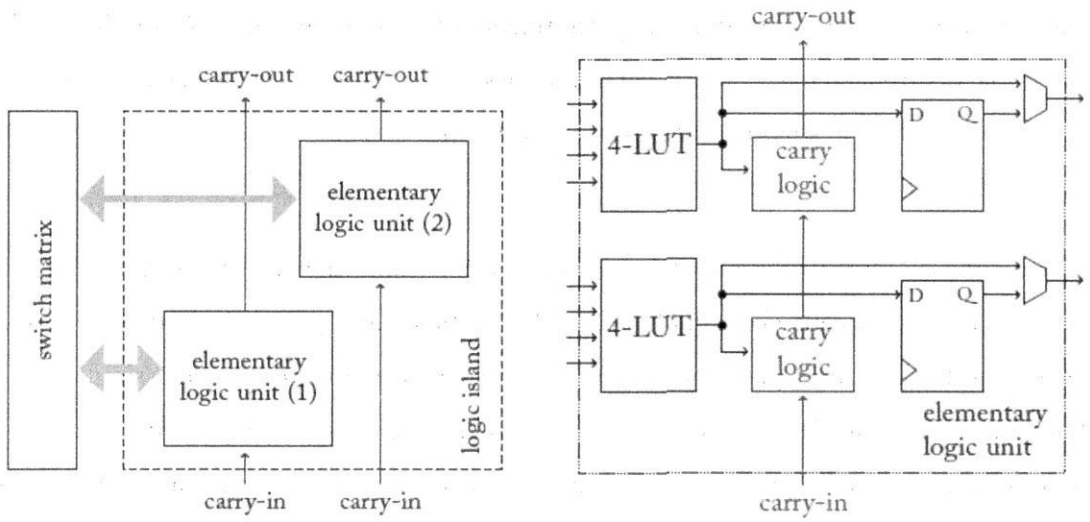


Figura 2.12 Organización de un bloque lógico configurable (CLB). a) cada CLB está formado por dos o más unidades elementales. b) cada unidad elemental contiene mínimamente un grupo reducido de LUTs, lógica de acarreo, biestables y multiplexores, se muestra la estructura clásica de una unidad lógica elemental de la familia Virtex 4 de Xilinx.

El elemento clave de una unidad lógica elemental son las tablas de búsqueda (LUT), pequeñas memorias de un bit de ancho que emula las expresiones booleanas del diseño, a diferencia de un ASIC en donde tales expresiones se implementan directamente mediante compuertas físicas. El principio es simple, una LUT de 2^n palabras puede implementar cualquier expresión de hasta *n* variables de entrada, basta con almacenar el resultado de la expresión para cada combinación de las variables de entrada en la celda apropiada de la memoria, luego utilizar las variables de entrada como dirección de la misma, lo anterior se muestra en la Figura 2.13a, donde se ha representado por simplicidad una LUT de 3 entradas, no obstante en la práctica son más comunes las LUTs de 6 entrada con circuitería sofisticada que permite convertirla a dos LUTs de 3 entradas si se requiere.

Bajo este principio una LUT puede ser (re)programada después de la manufactura, lo cual las hace responsables de la propiedad de (re)configurabilidad del FPGA. Físicamente cada LUT de n-entradas está compuesta de *n* biestables y un multiplexor de 2^n:1 (Figura 2.13b), a partir de esta estructura, cada LUT puede adquirir dos

funcionalidades adicionales, por una parte puede utilizarse como pequeños módulos de memoria distribuida para implementar bloques FIFO, LIFO, TLBs, etc. y por otra como registros de desplazamiento, para conversiones seriales-paralelo o como cadenas de retardo digital.

Dentro de la unidad lógica elemental cada LUT se aparea con un biestable (Figura 2.12b), el cual almacena el resultado de la tabla de búsqueda, permitiendo de esta forma el diseño de circuitos segmentados (una forma de procesamiento paralelo similar a las líneas de ensamble de una fábrica). Si el biestable se utiliza o no en un diseño se determina por el multiplexor de salida, cuyas líneas de selección se manejan a su vez por SRAM configurada durante el proceso de programación del FPGA. La lógica de acarreo por su parte, contiene circuitería especial que al combinarse con las LUTs pueden implementar módulos con funciones aritméticas, como sumadores y multiplicadores, las líneas de acarreo vertical permiten que múltiples unidades lógicas elementales se conecten en cascada para crear módulos aritméticos de mayor amplitud.

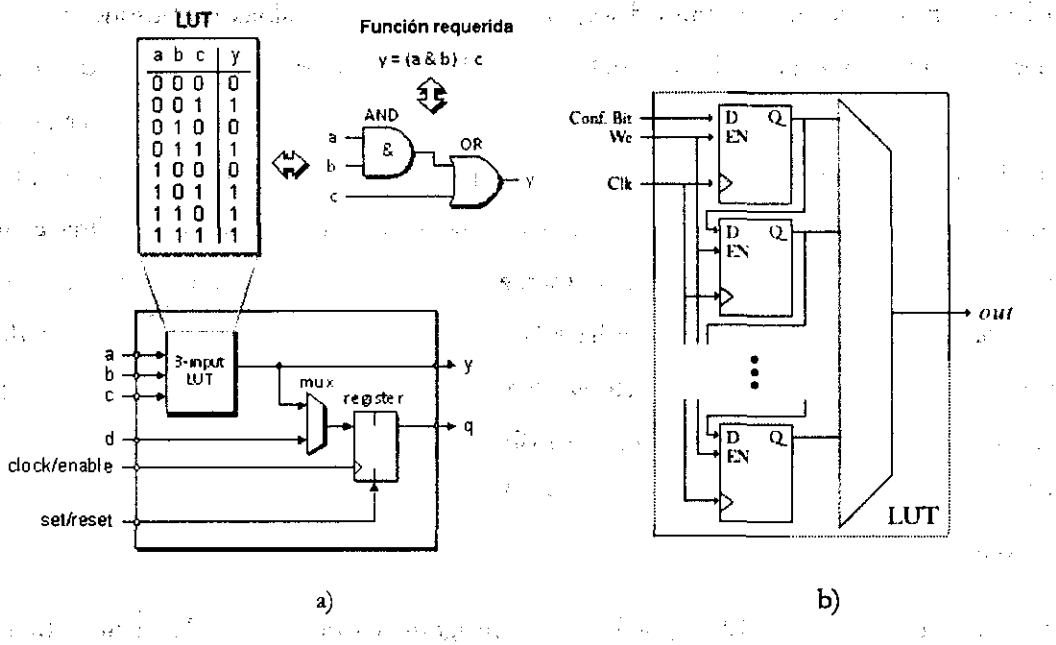


Figura 2.13 Tablas de búsqueda (LUT). a) Principio de funcionamiento: La tabla de verdad del circuito deseado se almacena en las localidades apropiadas de la LUT, luego las señales de entrada se utilizan como líneas de dirección. b) Organización interna: Cada LUT de n-entradas se forma de n biestables y un multiplexor de n a 1.

La red de interconexión es una arquitectura de ruteo configurable que permite la comunicación entre CLBs, ésta consiste de múltiples canales de comunicación (grupos de conductores de diferente amplitud y longitud) que corren vertical y horizontalmente a lo largo del chip, formando una malla conteniendo un CLB en cada punto de intercepción. En tales puntos existen enlaces programables, que determinan como los alambres se conectan entre ellos y por consiguiente como se conectan los diferentes CLBs del FPGA.

Finalmente, **los bloques de entrada y salida (IOBs),** rodean al arreglo bidimensional de CLBs. Estos IOBs situados en la periferia del FPGA también se conectan a la red de interconexiones permitiendo que la lógica interna se comunique con el exterior. Los IOBs pueden programarse para servir a diferentes necesidades y permitir al FPGA comunicarse con una multitud de dispositivos.

Típicamente se soportan muchos estándares de IO, tanto de salida única (usados por ejemplo en PCI) como de salida diferencial (usados por ejemplo en PCI express, SATA, 10G Ethernet, etc.). Adicionalmente los módulos IOBs pueden contener cierto hardware especial como convertidores seriales a paralelo, codificadores/decodificadores 8b/10b, etc., utilizados en diferentes protocolos de comunicación.

2.5.2.2 Componentes adicionales en el chip

La estructura básica del FPGA descrita en el apartado anterior, resulta en principio suficiente para implementar un rango amplio de circuitos. Sin embargo, para tratar las necesidades de usabilidad y alto desarrollo de algunas aplicaciones, los vendedores de FPGAs suelen incluir componentes especiales de silicio inmersos en la red de interconexión, tales como módulos dedicados de RAM (BRAM), multiplicadores y sumadores (unidades DSPs), y en algunos casos núcleos completos de Microprocesadores (Figura 2.14).

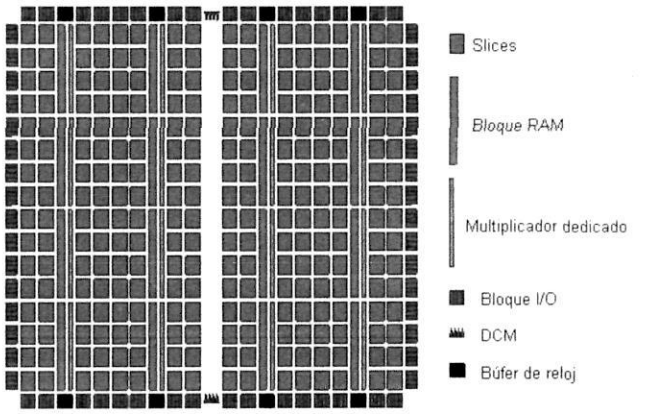


Figura 2.14 Un FPGA que incluye componentes adicionales en el chip.

Cada bloque BRAM tiene capacidad para unos cuantos kilobytes de datos (típicamente 4kiB) y usualmente un FPGA puede contener algunos cientos de módulos BRAMs, mismos que pueden accederse en paralelo. El acceso a BRAM es muy rápido, una palabra puede leerse o escribirse en un único ciclo de reloj a una velocidad de varios cientos de Megahertz. Comparado a la RAM distribuida los módulos BRAM proveen significantemente más alta densidad pero solo pueden instanciarse a una granularidad gruesa, haciéndolas la elección ideal para almacenar conjuntos de datos de uso inmediato en el chip. De igual manera que la RAM distribuida, pueden combinarse múltiples módulos BRAM para formar memorias más largas.

Otro bloque típico dentro del FPGA son los módulos de procesamiento digital (DSPs). Estos módulos consisten fundamentalmente de multiplicadores y sumadores implementados como componentes ASIC dentro del FPGA. Así estos componentes exhiben mayores prestaciones en términos de desarrollo, espacio y consumo de potencia que los que pudiesen haber sido logrados si se implementaran mediante LUTS, biestables y lógica de acarreo. Esta característica hace al FPGA especialmente atractivo para la implementación de funciones matemáticas, entre las que destacan el filtrado digital y el análisis de Fourier. Como con la mayoría de los otros componentes en el FPGA, las unidades DSP pueden personalizarse y combinarse con otras unidades adjuntas DSPs.

lógicas, seguida de la **tecnología de mapeo** quien separa las compuertas en grupos que mejor igualen a los recursos del hardware del FPGA. A continuación la **ubicación** asigna los grupos lógicos a bloques lógicos combinacionales y el **ruteo** determina los recursos de interconexión que portarán las señales del usuario. En el proceso es posible encontrar puntos de simulación posteriores a la síntesis y al ruteo que permiten depurar el diseño. Finalmente la **generación de la cadena de bits** creará un archivo binario, que contiene el código para configurar los bloques lógicos y recursos de ruteo apropiadamente.

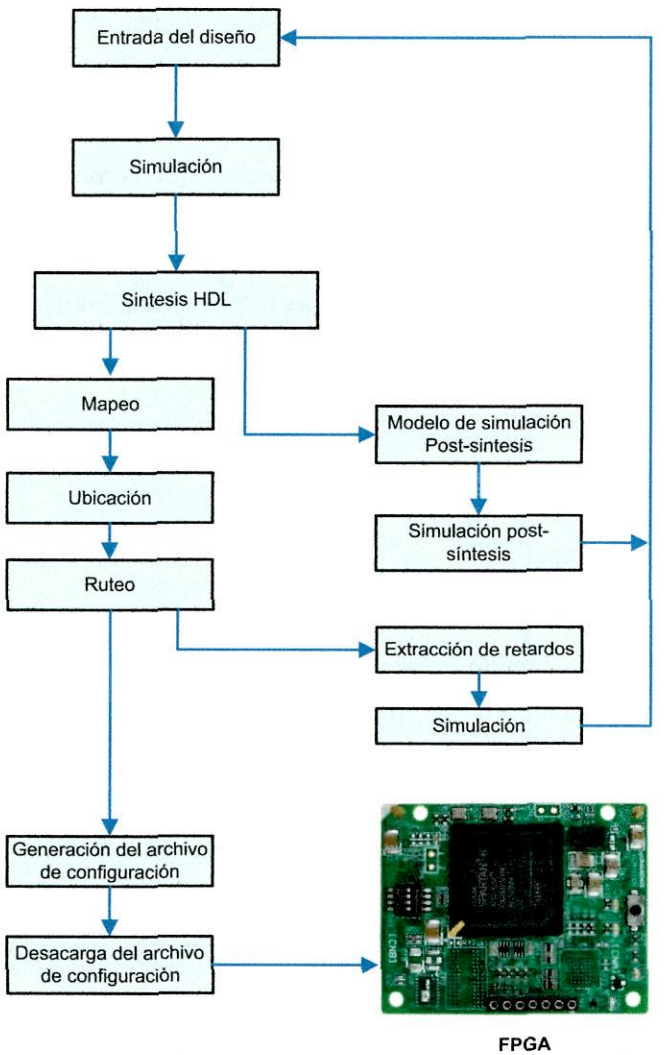


Figura 2.15 Diagrama de flujo de la implementación de un circuito o sistema dentro de un FPGA.

Además de lo anterior muchos fabricantes proporcionan los llamados núcleos con propiedad intelectual (IP), los cuales pueden instanciarse en los diseños FPGAs. En esencia, un núcleo IP es una unidad reutilizable de lógica (libre o comercial) que va desde operaciones aritméticas simples a microprocesadores completos. Un núcleo soft-IP implementa la funcionalidad correspondiente usando recursos programables estándar proporcionados por el FPGA, mientras que un Hard-IP refiere a un componente de silicio dedicado empotrado en la fábrica del FPGA. Los módulos BRAM y DSP son las formas más simples de componentes en silicio empotrados. Frecuentemente los vendedores agregan circuitería más compleja para soportar con alto desarrollo y mínimo consumo de espacio en el chip, por ejemplo, las nuevas familias de FPGAs de Xilinx de la serie Zynq incluyen procesadores de doble núcleo ARM Cortex-A9, mientras que Altera produce FPGAs con núcleos ARM empotrados, e Intel en colaboración con Altera diseñaron un procesador empotrado (Stellarton) que combina un núcleo Atom con un FPGA en el mismo paquete.

Esta generalidad y flexibilidad permite que en un FPGA pueda implementarse circuitos muy complejos e incluso sistemas completos en chip (SOC), equivalentes a millones de compuertas lógicas.

2.5.2.3 Configuración del FPGA

La implementación de un circuito o sistema dentro de un FPGA consiste básicamente en la creación de una cadena de bits para configurar el dispositivo (Figura 2.15), un proceso similar a compilar y cargar un programa en una computadora personal.

El proceso comienza introduciendo el diseño a la computadora mediante un lenguaje de descripción de hardware (por ejemplo VHDL o Verilog), un lenguaje de alto nivel (por ejemplo Handel-C o SystemC), un esquemático, u otro formato. Después de una simulación de alto nivel el diseño abstracto se optimiza para colocarlo dentro de la lógica disponible en el FPGA a través de una serie de pasos: La **síntesis lógica** convierte construcciones lógicas de alto nivel y código comportamental en compuertas

Después de que el diseño se ha compilado puede configurarse el FPGA para desarrollar un cálculo específico simplemente cargando el archivo binario dentro de este. Generalmente, lo anterior se logra ya sea mediante un microcontrolador/microprocesador quien descarga la cadena de bits al dispositivo o conectando una EEPROM programada con la cadena de bits al puerto de configuración del FPGA. De cualquier modo la cadena de bits debe cargarse cada vez que el FPGA se encienda, o cada vez que el usuario desee cambiar la circuitería en el caso de utilizar reconfiguración dinámica (Teubner & Woods, 2013).

2.6 Trabajos relacionados

Existen en la literatura varios trabajos que se han enfocado en acelerar los algoritmos de alineación de lecturas mediante hardware especializado, algunos de estos serán revisados a continuación.

SOAP3, presentado en (Liu C.-M. , y otros, 2012), es una presentación GPU habilitada por CUDA del popular software de alineación SOAP. Sus creadores reportan aceleraciones de 7.5X comparado a BWA, no obstante, su diseño no soporta alineación con espacios.

CUSHAW presentado en (Liu, Schmidt, & Maskell, 2012) es también una implementación basada en CUDA de los índices de FM, no permite alineaciones con espacios y reporta una aceleración de 6X comparado a la versión multi-hilos de BWA.

En (Nelson, Towsend, Rao, Jones, & Zambreno, 2012) los autores presentan SHEPARD una implementación FPGA de un algoritmo basado en tablas Hash sobre la plataforma de alto desarrollo Convey HC1. Reportan velocidades extremadamente altas comparadas con herramientas software como BWA, Bowtie y MAQ. No obstante, su arquitectura soporta únicamente alineación exacta y es capaz de alinear solo un bajo porcentaje de las lecturas entrantes, además de utilizar una elevada cantidad de memoria.

En (Fernández, Najjar, & Lonardi, 2011) se presenta la primer implementación de alineación exacta mediante los índices de FM. Al comparar sus resultados con el

software de alineación BOWTIE, obtienen aceleraciones mayores a 100X, sin embargo al almacenar la tabla de ocurrencia y el vector de frecuencia en módulos BRAM, su aplicación se limita a cadenas de ADN relativamente cortas.

Una mejora importante al trabajo de Fernández y equipo se reporta en (Arram, Tsoi, Luk, & Jiang, 2013), su método consiste de dos componentes claves, un alineador de lecturas exacto para el grueso del proceso de alineación y un alineador aproximado para los casos restantes. La arquitectura se implementa mediante configuración dinámica en el FPGA, maximizando el uso de recursos del mismo. Muestran que una implementación particular de su método utilizando un solo FPGA puede ser 293 veces más rápido que BWA sobre un procesador Intel X5650, y 134 veces más rápido que SOAP3 en una tarjeta NVIDIA GTX 580 GPU.

En (Waidyasooriya, Hariyama, & Kameyama, 2013), se presenta un acelerador hardware del software BWA, los autores reportan una aceleración de 10X. El diseño únicamente está probado en una referencia de pocos miles de pares bases, resultando en una velocidad de procesamiento menor que la utilizada usando referencias más grandes.

A excepción de SOAP3 y CUSHAW, el resto de los diseños son versiones preliminares y los resultados comparativos con alineadores software son estimaciones a partir de la razón de procesamiento.

3 PROGRAMAS Y ALGORITMOS DE ALINEACIÓN

El desarrollo acelerado de las tecnologías NGS ha provocado en los últimos años el surgimiento de gran cantidad de programas para la alineación de lecturas cortas a un genoma de referencia, debido a las características del problema, estos programas están soportados por algoritmos de elevada complejidad tanto temporal como espacial y diseñados para ejecutarse en computadoras con múltiples procesadores y/o múltiples núcleos. En este capítulo se analizan tales programas, su estructura general, características y un estudio a fondo de sus algoritmos de búsqueda principal.

3.1 Programas de alineación

En la actualidad existen más de 60 programas de alineación de material genómico (Fonseca, Rung, Brazma, & Marioni, 2012), (Shang, Zhu, Vongsangnak, Tang, Zhang, & Shen, 2014), (Ruffalo, LaFramboise, & Koyutürk, 2011), (Li & Homer, 2010), cada uno con diferentes características que lo hacen mejor en alguna aplicación específica (ADN, bisulfito, miARN, ARN, etc.) o plataforma de secuenciación (Illumina, Roche 454, etc.), algunos permiten alineación exacta, mientras otros no, también difieren en el número de errores permitidos e incluso en el sistema operativo en el cual se ejecutan. Las Tabla 3.1 resumen las características de los programas de alineación más representativos al momento de redactar este documento. En dicha tabla, la columna 2 hace referencia a las plataformas de secuenciación soportadas por el alineador codificadas de la siguiente manera: I para **I**llumina, S para ABI **S**olid, 4 para Roche **4**54, S para ABI **S**anger, H para **H**elicos, Ion para **Ion** Torrent y P para **P**acBio. Los límites en las longitudes de lecturas

se muestran en la columna tres, utilizando como unidad el nucleótido. La columna 4 y 5 indican si el alineador permite desapareos e *indels*, cuando es posible se ha registrado el número máximo permitidos de estos. La columna 6 indica si las inserciones y borrados consecutivos se permiten por el programa. En la columna 7 se muestra las alineaciones reportadas, en ésta se utiliza la nomenclatura: T-todas, M-La mejor, A-Aleatorio, U-Solo alineaciones únicas y S-número de diferencias definidas por el usuario. Las últimas columnas indican si el alineador utiliza información de calidad de las lecturas y su habilidad para manejo de lecturas por pares.

Tabla 3.1 Programas de alineación representativos. La nomenclatura se explica en el texto.

Programa de alineación	**Plataforma de Secuenciación**	**Longitud de lectura Min/Max**	**Desapareos permitidos**	**Inserciones y borrados Permitidos**	**Indels cons.**	**Alineaciones reportadas**	**Q A**	**P E**
Bfast	I,So,4,Hel	11/--	S	S	S	M,A,U	N	S
Bowtie	I,So,4,Sa,P	4/1k	S	S	N	T,M,A,S	S	S
Bwa	I,So,4,Sa,P	4/200	S	8	S	A,S	S	S
GASSST	I,So,4,Sa,P	50/500	S	S	N	A,M,U	-	-
Gmap	I,4,Sa,Hel,Ion,P	8/--	S	S	S	M	N	N
Maq	I,So	8/63	S	S	N	..	S	S
Novoaling	I,So,4,Ion,P	30/300	8	2	N	T,M,A,U,S	S	S
Pass	I,So,4	23/1K	S	S	S	T,M	S	S
Rmap	I.So,4	11/10K	S	0	N	M,S	S	S
Seqmap	I	15/500	5	3	N	T	N	N
Shrimp2	I,So,4	30/1K	S	S	N	T,M,R	N	S
Soap	I	7/60	5	3	N	M,A,U	N	S
Soap2	I	27/1k	2	0	S	T,M,A	N	S
Ssaha2	I,4,Sa	15/48K	S	S	N	M,S	N	S
Zoom	I,So,4	12/240	S	S	N	M,S,U	S	S

Otros dos aspectos de suma importancia son la velocidad de procesamiento y el uso de memoria RAM que requieren. En (Li & Homer, 2010) los autores realizan la comparación de algunos programas de alineación actuales, mediante el mapeo de 1 millón de lecturas al genoma humano en su versión g18, utilizando la configuración

default de cada software. La Tabla 3.2 muestra los programas más eficientes en tales aspectos de acuerdo a su experimento, como puede observarse, BFAST y Soap2 son los programas más rápidos, utilizando solo 39 y 97 minutos respectivamente, mientras que Bowtie2 y BWA son los que requieren menores recursos de memoria (5.1 y 7.6 Gb respectivamente). Estos valores son muy elevados si consideramos que en un proyecto de re-secuenciación suele requerirse la alineación de vacíos cientos de millones de lecturas.

Tabla 3.2 Comparación de programas de alineación. Fuente (Li & Homer, 2010).

Alineador	Tiempo total (min)	Tiempo de pre-procesamiento (min)	Tiempo de alineación (min)	Uso de memoria (GiB)	Tiempo/nt
BFAST	39	20	20	21.4	12 us
Bowtie2	176	160	16	5.1	9.6 us
BWA	97	83	14	7.6	8. 4 us
Soap2	82	70	12	7.8	7.2 us

3.2 Aproximación algorítmica

Algorítmicamente la alineación de secuencias puede lograrse a través de alguna de las siguientes estrategias:

1. Un algoritmo nativo, en el cual se escanea el genoma de referencia (R) para cada lectura (S_i), comparando la lectura en cada posición (p) y seleccionando el mejor apareo, aunque simple en concepto, la complejidad temporal es muy alta: $O(ml|R|)$, la cual hace a esta estrategia impráctica para cadenas relativamente largas.

2. El algoritmo de Knuth-Morris-Pratt (Knuth, Morris, & Pratt, 1977), el cual es una solución menos nativa, sin embargo su complejidad espacial aún sigue siendo elevada: $O(ml+m|R|)$.

3. Crear un árbol de sufijos para R y posteriormente realizar una búsqueda en el árbol para cada S_i. Si se asume que el árbol es construido mediante un algoritmo lineal en tiempo, la complexidad temporal llega a ser: $O(m/ + |R|)$, un valor práctico para los parámetros involucrados en la aplicación. Adicionalmente, esta solución tiene la ventaja de que el árbol debe construirse solamente una vez, guardado y usado repetidamente para mapear nuevas secuencias, al menos hasta que una versión actualizada del genoma se publique. No obstante su complejidad espacial viene a ser la principal limitante, esto considerando que cada hoja del árbol debe almacenar los índices donde el sufijo comienza, lo cual requiere $O(|R|\log|R|)$ bits solo para la codificación binaria de los índices, comparada con $|R|\log|\Sigma|$ bits para el texto original. Lo anterior se traduce en aproximadamente 64GB solo para almacenar el árbol del genoma humano, cuando el texto del mismo ocupa solo 750 MB. Claramente tal requerimiento supera la memoria cache de cualquier computadora personal contemporánea.

4. La cuarta solución consiste en pre-procesar un genoma de referencia en una tabla Hash H, la clave de los Hash son todas las sub-cadenas de longitud l en R, y el valor de cada clave es la posición p en R donde la sub-cadena inicia. Entonces dada S_i el algoritmo retorna $H(S_i)$. La complexidad temporal del método es muy buena: $O(m/+l|R|)$, sin embargo la complexidad espacial permanece muy alta: $O(l|R|+|R|log|R|)$, ya que debe mantenerse la representación binaria de la posición de cada subcadena. Para subsanar lo anterior suele trabajarse a nivel bit, representando cada nucleótido como un código de 2-bits, logrando una compactación con un factor de cuatro.

Debido a sus ventajas la mayoría de los programas de alineación de lecturas cortas utilizan solos dos estrategias: Tablas Hash y Transformada de Burrows-Wheeler (TBW), esta última siendo el resultado de la evolución de los árboles y arreglos de sufijos. Ambas estrategias serán discutidas en los apartados siguientes.

3.2.1 Algoritmos basados en Tablas Hash

Para lograr eficiencia, todos los métodos deben basarse en un tipo de pre-computo. Tablas Hash utiliza la idea de compilar una lista de todas las palabras de longitud *l* y determinar una sola vez sus posiciones en el genoma de referencia. Posteriormente puede utilizarse un algoritmo de hasheo, para transformar una lectura corta en una clave que permita una búsqueda rápida. Aunque esta idea es teóricamente concebible, falla en la práctica por el uso excesivo de memoria en la computadora, además de que, el esquema básico aún no considera alineaciones inexactas.

Una posible solución a este problema es el uso de *k-mers* (sub-cadenas de longitud *k*, con $k<l$), eligiendo un valor de *k* mucho menor a *l* se pueden almacenar todo los *k-mers* traslapados, que aparezcan en la referencia en una lista, tal como se observa en la parte a de la Figura 3.1. Posteriormente cada lectura puede dividirse en semillas de longitud *k* y buscarse sobre la lista (ver Figura 3.1, parte b). Si las semillas de una lectura se encuentran en las lista, en el orden correcto y adjuntas una a otra, la lectura existe en el genoma (parte c y d de la Figura 3.1). En términos del espacio utilizado, ahora el problema es más tratable ya que hay a lo mucho 4^k diferentes *k-mers* en el genoma. Aun así este algoritmo no permite alineación inexacta.

El algoritmo previo puede modificarse para alinear lecturas permitiendo errores (desapareos e indels). Suponga que se permiten dos errores durante la alineación, en tal caso se puede asegurar que a lo mucho dos de los *k-mers* en que se ha dividido la lectura contendrán errores y el resto de estos no los contendrán, alineándose en forma exacta al genoma (ver Figura 3.2), lo anterior se conoce como el principio de las cajas o del palomar. Los k-mer que se alinean perfectamente al genoma constituyen una "semilla", y al aplicar un algoritmo más preciso como los basados en programación dinámica en la vecindad de esta semilla, es posible alinear la lectura conteniendo errores. Esta estrategia de dos pasos llamada "siembra y extiende" se implementa en muchas herramientas tales como MAQ (Li, Ruan, & Durbin, 2008), PASS (Campagna, y otros, 2009), SSAHA2 (Ning, Cox, & Mullikin, 2001) , SOAP (Li, Li, Kristiansen, & Wang, 2008), RMAP (Smith, Xuan, & Zhang, 2008) y SeqMap (Jiang & Wong, 2008).

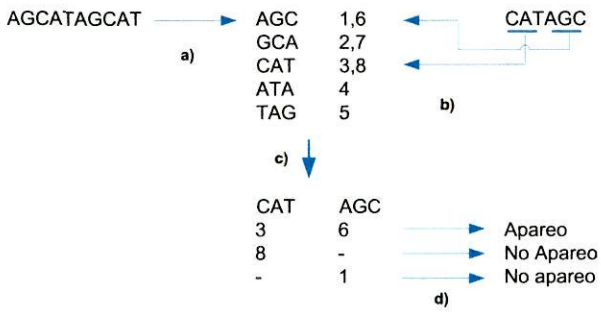


Figura 3.1 El algoritmo de hasheo. a) El genoma se divide en 3-mers traslapados y la posición de cada uno de estos se almacena en la tabla. b) La lectura también se dividen en semillas de tamaño 3 y se buscan en la tabla Hash. c) Las posiciones para cada semilla se comparan unas a otras c) Se obtienen las posiciones adjuntas y en el orden correcto, las cuales representan una alineación exacta.

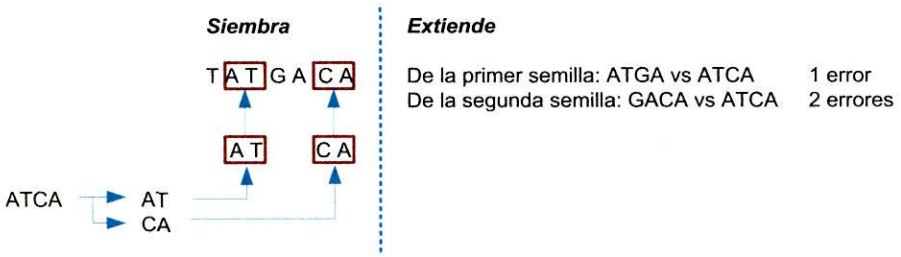


Figura 3.2 El algoritmo siembra y extiende. En el ejemplo, la lectura ATCA se busca en TATGACA permitiendo un error y usando semillas de tamaño 2. En la fase de siembra, cada semilla se alinea en una posición. En la extensión se observa que solo una de las alineaciones contiene un solo error.

Alternativamente, las lecturas también pueden ser hashedas, en cuyo caso se busca igualar regiones del genoma, con longitud igual a las lecturas. Sin embargo, ya que los secuenciadores de hoy en día son capaces de producir cantidades de lecturas enormes en una sola corrida, la solución demanda demasiada memoria y ha dejado de ser práctica.

El problema fundamental del algoritmo *siembra y extiende* es que las lecturas necesitan dividirse en subcadenas cada vez más pequeñas cuando el número de errores permitidos

se incrementa. Estas subcadenas tan pequeñas suelen producir errores en la fase del sembrado, puesto que tienen mayor probabilidad de alinearse equivocadamente en múltiples regiones del genoma (falsos positivos), es por esta razón que no es común el uso de semillas de menos de 10 nucleótidos. Para sobrellevar el problema, algunos programas como ZOOM (Lin, Zhang, Zhang, Ma, & Li, 2008), GASSST (Rizk & Lavenier, 2010), BFAST (Homer, Merriman, & Nelson, 2009) y SHRiMP2 (David, Dzamba, Lister, I.Le, & Brudno, 2011) han recurrido al uso de semillas espaciadas, es decir semillas conteniendo posiciones "no importa", en las cuales el algoritmo no checa el tipo de nucleótido presente. Por ejemplo, indicando a x como la posición no importa, la semilla AGTCxGA es capaz de alinearse a AGTCAGA, AGTCCGA, etc. Es evidente que utilizar un conjunto de semillas espaciadas en lugar de una semilla regular incrementa la sensibilidad del método, aunque tiene el efecto lateral de incrementar el tiempo de cómputo.

Las herramientas basadas en el algoritmo siembra y extiende gastan la mayor parte del tiempo en la etapa de extensión, implementada usualmente mediante algún algoritmos de programación dinámica, como el Smith-Waterman (Smith & Waterman, 1981) o el Needleman-Wush (Needleman & Wunsch, 1970), los cuales permiten en forma natural alineaciones locales con indels y desapareos. Para superar la complejidad temporal de esta etapa, los programas utilizan optimizaciones comunes de los algoritmos de programación dinámica y en algunos casos, como en SHRiMP2, comandos especiales del CPU para paralelizar el trabajo. Otros métodos agregan un paso intermedio entre la siembra y la extensión, tal es el caso de GASSST, el cual cada vez que encuentra una semilla, compara rápidamente la región vecina con el resto de la lectura, usando un algoritmo mucho más rápido que los de programación dinámica. La etapa agregada denominada *filtro*, consiste en calcular la distancia de Euler, la cual halla el número de letras de cada tipo en las regiones comparadas, si por ejemplo se trata de alinear una región conteniendo tres As, con una región que contiene cinco As, es evidente la existencia de al menos dos errores en la alineación, de esta forma las regiones que no cumplan con este filtro pueden ser descartadas disminuyendo notablemente la carga de la etapa de alineación. Esta variante del algoritmo se llama apropiadamente *siembra, filtra y extiende.*

3.2.2 *Algoritmos basados en TBW*

La transformada de Burrows-Wheeler (TBW), presentada originalmente en (Burrows & Wheeler, 1994), es un algoritmo que transforma una cadena de caracteres en otra mucho más fácil de comprimir. El re-ordenamiento se hace en tal modo que la cadena resultante agrupa los caracteres similares en la cadena original, de esta forma puede ser comprimida fácilmente usando codificación *move-to-front* y codificación *run-length*. No obstante, recientemente la TBW ha encontrado una aplicación diferente a la compresión de datos, luego de que Ferragina y Manzini (Ferragina & Manzini, 2000) mostraron como la TBW hereda todas las propiedades y potencia computacional de los arreglos de sufijos, pero requiriendo mucho menos espacio en memoria que estos. De esta forma, numerosos programas de alineación recientes como SOAP2 (Li, y otros, 2009), Bowtie (Langmead, Trapnell, Pop, & Salzberg, 2009) y BWA (Li & Durbin, 2009), utilizan la TBW como estructura de datos principal.

3.2.2.1 *Transformada de Burrows-Wheeler*

Más que establecer una definición rigurosa de la TBW, en lo que sigue se presenta el algoritmo mediante el ejemplo simple mostrado en la Figura 3.3.

Sea R=TAGATACA la cadena de entrada a transformar, Σ={A,C,G,T}, el alfabeto sobre el cual es construida la cadena, y \$ un carácter especial lexicográficamente menor que todos los otros caracteres en Σ. El algoritmo comienza agregando el carácter especial \$ a la cadena de entrada R, de esta forma obteniendo la secuencia R'=TAGACAGA\$, el carácter especial es necesario para poder revertir la transformación y mantener la compatibilidad con el arreglo de sufijos equivalente, como se verá en el capítulo siguiente. Posteriormente se generan todas las rotaciones de R' y se colocan en un arreglo conceptual como se muestra en la Figura 3.3a, observe como en tal arreglo pueden identificarse todos los sufijos posibles de la cadena R. Finalmente las rotaciones se ordenan lexicográficamente como se muestra en la Figura 3.3b. La última columna del arreglo es la transformada buscada, en el ejemplo:

$L(i)$=AGGCTAAA$. En los arreglos de la Figura 3.3 las variables i y $S(i)$ representan el índice de los arreglos, y el arreglo de sufijos correspondientes, ambas colocadas solamente por claridad.

Note como aun a pesar de que el uso asintótico de memoria es $O(|R|)$, la cantidad de almacenamiento que se requiere para guardar la TBW es significantemente más pequeño que el necesario para un arreglo de sufijos. Almacenar la TBW requiere el mismo espacio que el texto original, ya que esta es solo una permutación de dicho texto. Adicionalmente, en el caso que nos concierne, el alfabeto consiste de solo cuatro letras, de tal forma que cada una de estas puede representarse mediante dos bits, así para almacenar la TBW del genoma humano se requiere $\sim 2*3*10^9$ bits, en lugar de los $\sim 32*3*10^9$ bits del arreglo de sufijos del mismo, suponiendo que cada entero utiliza 32 bits, como en la mayoría de las aplicaciones.

i									
0	T	A	G	A	C	A	G	A	$
1	$	T	A	G	A	C	A	G	A
2	A	$	T	A	G	A	C	A	G
3	G	A	$	T	A	G	A	C	A
4	A	G	A	$	T	A	G	A	C
5	C	A	G	A	$	T	A	G	A
6	A	C	A	G	A	$	T	A	G
7	G	A	C	A	G	A	$	T	A
8	A	G	A	C	A	G	A	$	T

a)

i	S(i)								L(i)
0	1	$	T	A	G	A	C	A	G	A
1	2	A	$	T	A	G	A	C	A	G
2	6	A	C	A	G	A	$	T	A	G
3	4	A	G	A	$	T	A	G	A	C
4	8	A	G	A	C	A	G	A	$	T
5	5	C	A	G	A	$	T	A	G	A
6	3	G	A	$	T	A	G	A	C	A
7	7	G	A	C	A	G	A	$	T	A
8	0	T	A	G	A	C	A	G	A	$

b)

Figura 3.3 La transformada de Burrows-Wheeler de la cadena R= TAGACAGA. a) Después de agregar el carácter especial a la cadena R, se crean todas sus rotaciones escribiéndolas en una matriz conceptual. b) las filas de la matriz se ordenan lexicográficamente, siendo la última columna el resultado de la transformación.

La transformada de Burrows Wheeler es un proceso reversible, lo que significa que existe un procedimiento mediante el cual es posible recuperar el texto original a partir del resultado de la transformación. El algoritmo de reconstrucción lineal en tiempo fue

presentado en (Burrows & Wheeler, 1994) y está basado en la propiedad conocida como mapeo Ultimo-Primero (LF), enunciada a continuación:

Lema: (Propiedad de mapeo Ultimo-Primero) *La j-ésima ocurrencia de una letra en particular X en la última columna (L) de la matriz TBW, es la misma letra como la j-ésima ocurrencia de X en la primera columna (F).*

Comprobación: Sea X_j la *j*-ésima ocurrencia del carácter X en L, α el carácter que sigue a X y β el carácter que sigue a X_{j+1} ambos en la cadena original R. Entonces, ya que X_j aparece arriba de X_{j+1} en L, a debe ser igual o lexicográficamente más pequeño que β. Esto es cierto ya que los caracteres en F siguen en una posición a los caracteres en L en cada fila y cada columna se encuentra ordenada lexicográficamente. De aquí que, cuando el carácter X_j aparezca en F, este nuevamente estará arriba de X_{j+1}, ya que a y β, ahora aparece en la segunda columna y $Xa \leq X\beta$.

Informalmente, para recuperar la cadena original R a partir de su transformada, se hace referencia a la primera y la última columna de la matriz TBW (F y L respectivamente), las cuales para el caso del ejemplo previo se muestran en la Figura 3.4. En esta Figura cada subíndice indica la posición relativa de aparición en la columna respectiva, también conocida como rango del carácter, el símbolo $ no fue etiquetado, puesto que aparece siempre en una sola ocasión.

El procedimiento comienza identificando el primer carácter en la columna L, el cual en este caso es A_1., mismo que se intuye ser el último en la cadena original, al aparecer a la izquierda del símbolo $ (Recuerde que al ser cada fila una rotación del texto, cada carácter en L precede al carácter en F en esa misma fila). Por la propiedad del mapeo LF, ese mismo carácter se encuentra en la fila 1 de la columna F, y es precedido a la izquierda por el carácter G_1 (último carácter de esa misma fila). G_1 a la vez encuentra su mapeo en la posición 6 de F y puede notarse que es precedido por A_3. El procedimiento continúa de esta manera hasta alcanzar el carácter especial, al hacerlo se tiene la cadena original recuperada de derecha a izquierda $T_1A_4G_2A_2C_1A_3G_1A_1\$$.

i	F		L
0	$	.....	A_1
1	A_1	.....	G_1
2	A_2	.....	G_2
3	A_3	.....	C_1
4	A_4	.....	T_1
5	C_1	.....	A_2
6	G_1	.....	A_3
7	G_2	.....	A_4
8	T_1	.....	$

Figura 3.4 La primera y la última columna de la matriz TBW etiquetados de acuerdo a la propiedad del mapeo Ultimo-Primero.

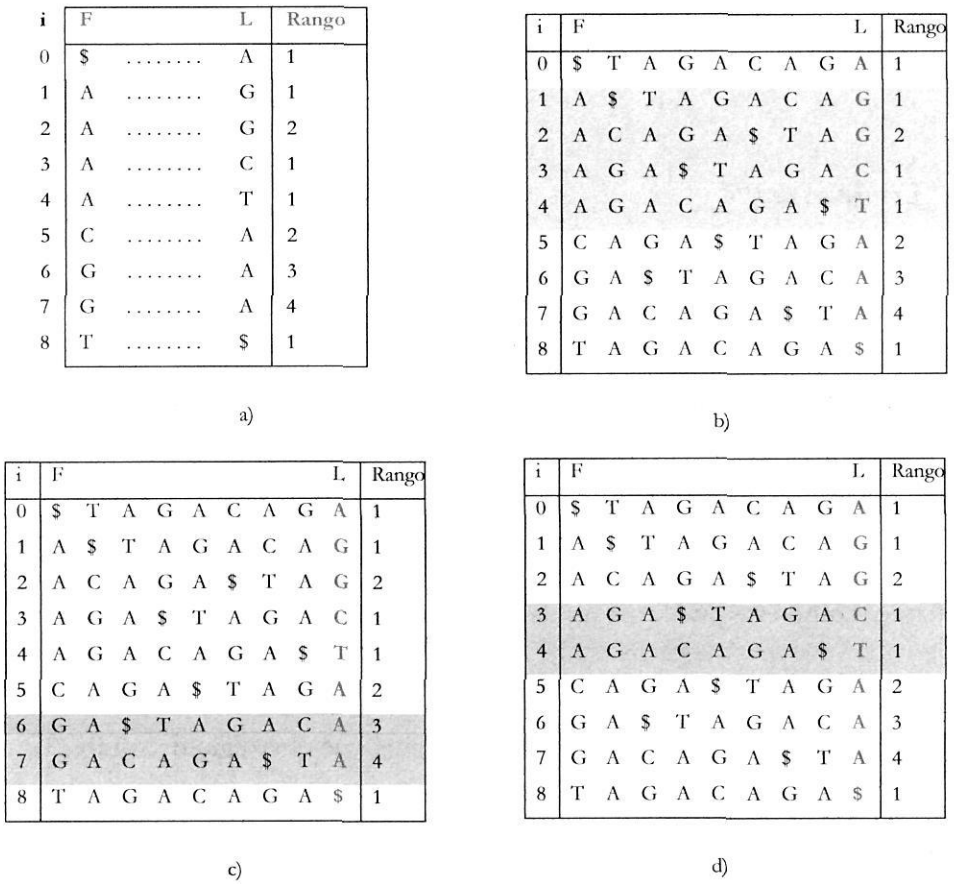
3.2.2.2 Los índices de FM

Seis años después de que la TBW fuera publicada, Paolo Ferragina y Geovanni Manzini publicaron un artículo (Ferragina & Manzini, 2000) en el que se describía como la TBW, junto con algunas estructuras de datos auxiliares, podría usarse como un índice de R eficiente en espacio, a lo que llamaron índices de FM. Estas estructuras generalmente usan menos de la mitad del espacio requerido para almacenar $|R|$ enteros. Así como el mapeo LF fue la clave para entender como la TBW es reversible, también es la clave para entender como esta puede ser usada como un índice.

Suponga que se desea buscar las ocurrencias del patrón P=AGA en la cadena de referencia R del ejemplo anterior. Puesto que al igual que el arreglo de sufijos, la matriz TBW esta ordenada lexicográficamente, las filas que contienen a P como un prefijo se encontrarán de manera consecutivas.

El procedimiento de búsqueda se basa en el índice formado por F, L y un vector que contiene el rango de los caracteres en L denominado apropiadamente vector de rango (Figura 3.5a). La búsqueda inicia, encontrando las filas que comienzan con el sufijo más corto de P, A en este caso, lo anterior es trivial puesto que la primera columna es parte del índice, estas filas son aquellas en el intervalo [1, 4] (Figura 3.5b).

A continuación puede agregarse el siguiente carácter a la búsqueda formando el segundo sufijo más corto del patrón: GA. El espacio de búsqueda ahora está limitado al intervalo [1,4] hallado previamente. Observando la columna *L* en ese rango puede determinarse que existen dos filas en las cuales G precede a A (filas 1 y 2) y al aplicar el mapeo LF utilizando el vector de rango puede hallarse que esos mismos caracteres se localizan en las filas 6 y 7 de *F*, en otras palabras, se ha obtenido un nuevo intervalo de búsqueda: [6,7], como se muestra en la Figura 3.5c.



a)

i	F	L	Rango
0	$	A	1
1	A	G	1
2	A	G	2
3	A	C	1
4	A	T	1
5	C	A	2
6	G	A	3
7	G	A	4
8	T	$	1

b)

i	F	L	Rango
0	$ T A G A C A G	A	1
1	A $ T A G A C A	G	1
2	A C A G A $ T A	G	2
3	A G A $ T A G A	C	1
4	A G A C A G A $	T	1
5	C A G A $ T A G	A	2
6	G A $ T A G A C	A	3
7	G A C A G A $ T	A	4
8	T A G A C A G A	$	1

c)

i	F	L	Rango
0	$ T A G A C A G	A	1
1	A $ T A G A C A	G	1
2	A C A G A $ T A	G	2
3	A G A $ T A G A	C	1
4	A G A C A G A $	T	1
5	C A G A $ T A G	A	2
6	G A $ T A G A C	A	3
7	G A C A G A $ T	A	4
8	T A G A C A G A	$	1

d)

Figura 3.5 Búsqueda del patrón P=AGA en la cadena de referencia R=TAGACAGA. a) El índice está formado por el vector de rango, la primera (F) y la última (L) columna de la matriz TBW. b) Intervalo hallado al considerar el sufijo más corto del patrón: A. c) Nuevo intervalo al extender el sufijo: AG. d) Intervalo final al extender nuevamente el sufijo: AGA.

Posteriormente se agrega el último carácter del patrón, formando el sufijo AGA. Nuevamente revisando *L* en el intervalo encontrado, puede observarse que existen dos filas en las cuales A precede al prefijo GA (filas 6 y 7), y al aplicar el mapeo LF se halla que esos mismos caracteres en *L* se encuentran en las posiciones 3 y 4 respectivamente, el cual efectivamente es el intervalo en el que el prefijo buscado aparece (Figura 3.5d).

Este procedimiento se denomina búsqueda hacia atrás. En resumen, consiste en aplicar el mapeo LF repetidamente para encontrar el rango de filas prefijadas por sufijos de *P* alargados progresivamente, hasta que el rango llegue a estar vacío, lo cual ocurre si *P* no existe en *R*, o hasta que se termine con los sufijos, en cuyo caso el tamaño del rango es el número de veces que *P* ocurre en *R*. Observe como en la Figura 3.5 se incluyó la matriz TBW completa solo por claridad, interviniendo en este ejemplo inicial solo la primera y última columna de la misma.

La complejidad temporal del mismo puede mejorarse si en lugar de almacenar un vector de rangos se almacena la matriz de ocurrencia *Occ*, en cada fila de tal matriz se guarda un entero para cada carácter del alfabeto igual al número de veces que el carácter ha aparecido en *L*, hasta ese número de fila e incluida esta, lo anterior puede observarse en la Figura 3.6. Ahora en lugar de escanear la última columna, simplemente se busca el carácter apropiado en los extremos izquierdo y derecho del rango actual, si no hay diferencias entre las dos búsquedas, el carácter no ocurre. Si hay una o más ocurrencias del carácter, la búsqueda dará los rangos de esas ocurrencias.

			Matriz de Ocurrencias			
i	F	L	A	C	G	T
0	$	A	1	0	0	0
1	A	G	1	0	1	0
2	A	G	1	0	2	0
3	A	C	1	1	2	0
4	A	T	1	1	2	1
5	C	A	2	1	2	1
6	G	A	3	1	2	1
7	G	A	4	1	2	1
8	T	$	4	1	2	1

Figura 3.6 Índice de la secuencia de referencia R=TAGACAGA incluyendo la Matriz de ocurrencias (O)

El algoritmo de búsqueda formalizado, adaptado de la referencia (Ferragina & Manzini, 2000), se presenta en el Listado 3.1. En este interviene la matriz de ocurrencias *Occ* previamente definida y el vector de frecuencias *C* de longitud $|\Sigma|$ en donde *C*['x'] representa el número de caracteres en *R* que son lexicográficamente más pequeños que 'x' sin considerar el carácter especial $. En el ejemplo particular que se ha seguido este corresponde a:

	A	C	G	T
C =	0	4	5	7

Listado 3.1 Algoritmo para realizar búsquedas exactas usando los índices de FM

```
1   function EXACTMATCH (R, C, Occ)
2   i ← |P|-1
3   k ←0
4   l ←|R|- 1
5   while k ≤ l ∩ i ≥ 0 do
6   σ← P [i]
7   k← C[σ] + Occ[σ,k-1 ] + 1
8   l ← C[σ] + Occ[σ,l ]
9   i ← i- 1
10  end while
11  if k ≤ l then
12  return {k,l}
13  else
14  return {Φ}
15  end if
16  end function
```

Las variables *k* y *l* representan los extremos actuales del intervalo inferior y superior respectivamente, *P* el patrón buscado, *i* un apuntador y σ una variable que contiene el carácter del patrón procesado actualmente. Las líneas 2 a 4 inicializan el proceso colocando el índice *i* apuntando al carácter más a la derecha del patrón y los índices *k* y *l* indican el intervalo más grande posible de búsqueda (0 y $|R|-1$, respectivamente).

Las líneas 5 a la 10 forman el núcleo del proceso, básicamente el bucle toma uno a uno los caracteres del patrón de derecha a izquierda, aplicando el mapeo último a primero en

las líneas 7 y 8 para definir el nuevo intervalo de búsqueda, tal como se explicó previamente. Observe como el término $C[\sigma] + 1$, en realidad señala la primer posición de aparición del carácter σ en la primera fila y se suma al termino Occ[σ,k-1], que representa el número de ocurrencias del mismo carácter hasta la fila k-1, para obtener el mapeo deseado. Finalmente el algoritmo retorna el intervalo $[k,l]$ si $k \leq l$ o un intervalo vacío en caso contrario (líneas 10 a 15).

La complejidad temporal del algoritmo es $O(|P|)$, el hecho de que la complejidad en tiempo no dependa del tamaño de la cadena de referencia es sumamente ventajoso para este tipo de aplicaciones, en donde secuencias cortas se alinean a cadenas extremadamente largas tales como el genoma humano.

En la Tabla 3.3, se muestran los resultados en cada paso del algoritmo al buscar el patrón P=AGA en la cadena de referencia R = TAGACAGA. Note como el algoritmo retorna los índices de las filas en la matriz de transformación que comienzan con el patrón pero no provee la ubicación de cada concordancia en R. La forma inmediata de obtenerla es a través del arreglo de sufijos, sin embargo almacenarlo completamente requiere demasiada memoria, en su lugar solo se guarda un grupo de filas con posiciones pre-calculadas denominadas "marcas". Entonces, si el algoritmo retorna un valor correspondiente a una marca, se toma ese valor inmediatamente, en caso contrario, debe aplicarse sucesivamente el mapeo LF hasta encontrar otra marca. La suma del valor de la marca original con el número de veces que se aplicó el mapeo LF, dará la ubicación deseada.

Tabla 3.3 Resultados de la búsqueda del patrón P=AGA, en la referencia R=TAGACAGA

Iteración	i	σ	K	L
Inicio	2		0	8
1	2	A	$k \leftarrow C[A] + Occ[A,-1\ ] + 1=1$	$l \leftarrow C[A] + Occ[A,8\ ] =4$
2	1	G	$k \leftarrow C[A] + Occ[A,-1\ ] + 1=6$	$l \leftarrow C[A] + Occ[A,8\ ] =7$
3	0	A	$k \leftarrow C[A] + Occ[A,-1\ ] + 1=3$	$l \leftarrow C[A] + Occ[A,8\ ] =4$
Retorno			3	4

El esquema que acaba de presentarse no toma en cuenta búsquedas aproximadas. No obstante es posible adaptar el algoritmo para conseguirlo, habitualmente lo anterior se logra mediante uno de los siguientes métodos:

1. Expresando la búsqueda aproximada en términos de muchas búsquedas exactas. La idea básica es crear la vecindad de palabras del patrón *P*, que se encuentren a una distancia de edición[2] *d* (siendo *d* el número de diferencias permitidas en la búsqueda). Una vez generada la vecindad simplemente se busca cada palabra sobre el índice y se guarda cada acierto como una concordancia aproximada en esa ubicación dentro del texto. Lo anterior puede sonar muy simple sin embargo, el tamaño de la vecindad crece abruptamente al incrementar el número de diferencias permitidas. El tamaño de la vecindad de una palabra fue limitado a $O(|P|^d|\Sigma|^d)$ en [43], así si se intenta crear la vecindad de una lectura corta de ADN con 32 bases nucleicas de longitud y permitiendo 3 diferencias, el número de palabras seria de 2 985 984. Debido a la explosión combinatoria, el método solo es práctico para valores pequeños de *d*.

2. Adoptando la estrategia siembra y extiende. La técnica es idéntica a la presentada en la sección 2.5.2. De acuerdo a los parámetros de la búsqueda algunas sub-cadenas del patrón son localizadas en forma exacta en el índice y sumadas a una lista de concordancias candidatas, las cuales son posteriormente examinadas alrededor de sus posiciones mediante un algoritmo más preciso, validando o no la concordancia del patrón completo.

El lector interesado puede encontrar en (Navarro, Baeza-Yates, Sutinen, & Tarhio, 2001), una descripción más detallada acerca de posibles modos de hallar concordancias aproximadas Debe notarse que, distinto a lo que pasa con tablas Hash, en tal esquema de búsqueda no se modifica el índice al permitir diferencias, la extensión es siempre realizada algorítmicamente, pero desarrollando una búsqueda más compleja.

[2] La distancia de Levenshtein o distancia de edición, es el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra. Entendiendo por operación, una inserción, una eliminación o la substitución de un carácter.

4 DISEÑO E IMPLEMENTACIÓN

En el capítulo anterior se analizaron los principales algoritmos de búsqueda que utilizan los programas de alineación actuales, también se mostró como una búsqueda inexacta, puede implementarse a partir de múltiples búsquedas exactas, lo que permitió concluir que acelerar la ejecución de tales algoritmos repercutirá de manera importante en el desempeño global del programa. Este capítulo se enfoca en discutir como el algoritmo de Ferragina y Manzini o Índices de FM, puede mapearse en hardware reduciendo notablemente los tiempos de ejecución. La elección del mismo está fundamentada en tres principales razones: su relativamente bajo uso de memoria RAM, la facilidad de adaptarlo a búsquedas inexactas sin modificar los índices y a sus operaciones lógicas y aritméticas las cuales son múltiples pero muy simples e ideales para implementarse en un dispositivo lógico programable.

4.1 Especificaciones del diseño

Antes de proponer una arquitectura para el alineador deben establecerse las especificaciones del diseño y discutir algunos aspectos importantes en relación a la implementación hardware. En general se desea:

- Diseñar e implementar un acelerador hardware del algoritmo de búsqueda exacta basado en los índices de FM.
- La capacidad del sistema será tal que permita la alineación de millones de lecturas con longitudes entre 35 y 64 nucleótidos a genomas de referencia tan grandes como el genoma humano (3 000 millones de nucleótidos).

- El diseño debe reportar todas las alineaciones exactas de una lectura, las inexactas quedan fuera del alcance de este trabajo.
- La meta fundamental es optimizar la razón de procesamiento, maximizando el número de lecturas procesadas por unidad de tiempo. El área es importante puesto que influye en el número de alineadores dentro de un FPGA y por consiguiente la velocidad. La latencia y la potencia no son objetivos en este momento.
- El sistema debe incluir algún mecanismo que permita corroborar el funcionamiento del acelerador comprobando que sus resultados son los mismos que los entregados con una función en software.

El principal problema con respecto a una posible implementación hardware del algoritmo de búsqueda exacta discutida en la sección 3.2.2.2, es que las interacciones dependen de los valores calculados en las interacciones previas. El algoritmo de búsqueda exacta no permite realizar cálculos intermedios de manera anticipada, así, es imposible pre-calcular el intervalo en el arreglo de sufijos en el cual una búsqueda termina antes de hacer la siguiente.

Otro inconveniente es la gran cantidad de espacio en memoria que se requiere para almacenar los índices de FM. En particular, un sistema con la capacidad descrita en la sección previa y suponiendo que los enteros se almacenan en un formato de 4 bytes, como en la mayoría de las computadoras, utilizaría $|\Sigma| \times |R| \times 32 \approx 48$ GiB, únicamente para almacenar el arreglo de ocurrencias. El arreglo de sufijos a su vez necesitaría $|R| \times 32 \approx 12$ GiB, mientras que el vector de frecuencias $|\Sigma| \times 4 = 16$ B.

El espacio para almacenar las lecturas también es un punto importante, por ejemplo en un proyecto de re-secuenciación típico se requiere alinear alrededor de 100 000 000 de lecturas, si cada una tiene una longitud de 64 nucleótidos se necesitarían aproximadamente 6.4 GiB adicionales en memoria, suponiendo que cada nucleótido es almacenado como un carácter.

4.2 Optimizando el uso de memoria

Los problemas de memoria discutidos previamente, pueden resolverse si en lugar de almacenar el arreglo de ocurrencias (*Occ*) en su totalidad, únicamente se almacenan ciertas filas de la misma, denominadas marcas, y recuperar el resto al momento de ser necesarias, lo anterior se logra contando los caracteres en el segmento apropiado de la TBW. Por ejemplo, en Figura 4.1a se muestra la matriz de ocurrencia del ejercicio que se ha tratado a lo largo del documento, y en la Figura 4.1b la matriz de ocurrencia reducida (*OccR*) de la misma, en esta última se ha almacenado una marca por cada cuatro filas de la matriz original, si por ejemplo, en alguna iteración el algoritmo necesita el valor de ocurrencia de la Adenina en la fila 6, se lee el valor de la Adenina en la marca inferior más próxima (marca 4 en este caso), el cual es 1, y se suma al número de Adeninas que aparecen en el segmento de la TBW comprendida en la posición de la marca 4 y la posición 6 de interés, el cual es 2, dando como resultado el valor 3, mismo que es el buscado como puede confirmarse al observar la matriz original en esa posición.

i	TBW	Matriz de Ocurrencias (Occ)			
		A	C	G	T
0	A	1	0	0	0
1	G	1	0	1	0
2	G	1	0	2	0
3	C	1	1	2	0
4	T	1	1	2	1
5	A	2	1	2	1
6	A	3	1	2	1
7	A	4	1	2	1
8	$	4	1	2	1

a)

i	TBW	Matriz de Ocurrencias Reducida (OccR)			
		A	C	G	T
0	A	1	0	0	0
1	G				
2	G				
3	C				
4	T	1	1	2	1
5	A				
6	A				
7	A				
8	$	4	1	2	1

b)

Figura 4.1 a) Matriz de ocurrencia (Occ), contrastada con b) Matriz de ocurrencia reducida (OccR).

Es evidente que al almacenar una de cada n filas en la matriz de ocurrencia se tiene una compactación en un factor n. Esta técnica es una de las principales aportaciones de

programas como Bowtie y BWA que les permite ejecutarse en computadoras de escritorio y laptops convencionales con mínimo uso de memoria RAM. En este trabajo se adopta la estrategia almacenando una de cada 64 filas requiriendo únicamente alrededor de 750 MiB para la *OccR* del genoma humano.

Adicionalmente tanto la TBW como las lecturas se codifican utilizando únicamente 2 bits por nucleótido, en lugar de un byte completo, logrando una optimización en el uso de memoria en un factor de 4 en ambas estructuras. Las bases nucleicas se codifican de la siguiente manera: 00 para la adenina, 01 para la Citosina, 10 para la Guanina y 11 para la Timina, mientras que el carácter especial $ no se almacena, requiriendo un módulo especial de corrección en el diseño, para evitar errores durante la alineación.

Los nuevos requisitos de memoria del sistema ahora son menores y se muestran en la Tabla 4.1. En esta, solo se consideran las estructuras que se enviarán a la memoria local de la tarjeta de aceleración.

Tabla 4.1 Requisitos de memoria del sistema de aceleración.

TBW	OccR	Vector de frecuencias	Total
≈750 MiB	≈750 MiB	16B	≈1.5 GiB

4.3 Maximizando el ancho de banda lógico

De acuerdo a la discusión previa, para recuperar el valor de ocurrencia de un nucleótido en específico, el algoritmo requiere en cada iteración de: 1) Los datos de la marca inferior inmediata a la posición buscada y 2) el segmento de la TBW comprendida entre las posiciones de las marcas inferior y superior más próximas a la de interés. Esto sugiere almacenar en posiciones consecutivas ambos datos y recuperarlos en un solo acceso a memoria, de esta manera maximizando el uso del ancho de banda lógico de la tarjeta de desarrollo y en consecuencia acelerando la aplicación. Lo anterior puede ejemplificarse en la *Figura 4.2*, en donde se han almacenado de manera apropiada las marcas y los segmentos de la TBW del problema que se ha utilizado como ejemplo.

0	1	0	0	0	A	G	G	C
1	1	1	2	1	T	A	A	A
2	4	1	2	1	$	X	X	X

Figura 4.2 Almacenamiento intercalado de *OccR* y TBW para optimizar la velocidad de procesamiento.

La técnica es conocida en el área de arquitectura de computadoras como intercalado de memoria, y se utiliza en el presente trabajo en virtud del elevado ancho de banda lógico de la tarjeta Pico Computing M505 k325T utilizada como soporte para el acelerador.

La tarjeta M505 k325T es capaz de transferir datos de hasta 256 bits (32 bytes) en una sola lectura, esto llevó a la elección del valor 64 como intervalo óptimo entre marcas, es decir, cada línea en el mapa conceptual de memoria puede almacenar 256 bits, los 128 bits más significativos contendrán los cuatro valores de una fila de la matriz de ocurrencia reducida mientras que los 128 bits menos significativos pueden almacenar segmentos de la TBW de hasta 64 nucleótidos, justo el tamaño del segmento de la TBW involucrada en cada calculo.

Lo anterior se representa en la Figura 4.3 para las primeras filas del mapa, en esta figura la letra 'X' representa un byte almacenando valores de ocurrencia y la letra 'Y' un byte almacenando caracteres de la TBW. Si por ejemplo se utilizará un valor inferior a 64 como el 32, se seguiría utilizando 128 bits para almacenar los valores del arreglo de sufijos pero se usarían solo 64 de los disponibles 128 bits menos significativos para almacenar los 32 nucleótidos del segmento, un sistema como tal haría un uso ineficiente del ancho de banda lógico.

```
00000000    XXXXXXXXXXXXXXXX YYYYYYYYYYYYYYYY
00000020    XXXXXXXXXXXXXXXX YYYYYYYYYYYYYYYY
00000040    XXXXXXXXXXXXXXXX YYYYYYYYYYYYYYYY
00000060    XXXXXXXXXXXXXXXX YYYYYYYYYYYYYYYY
00000080    XXXXXXXXXXXXXXXX YYYYYYYYYYYYYYYY
000000A0    XXXXXXXXXXXXXXXX YYYYYYYYYYYYYYYY
```

Figura 4.3 Primeras filas del mapa conceptual de memoria de la DRAM en la tarjeta M505 k325T, al utilizar un valor entre marcas de 64.

Es importante en este punto discutir la manera de encontrar los datos necesarios para el cálculo de un valor de ocurrencia solicitado por el algoritmo. Para esto k (o l) debe interpretarse como formado por dos partes, los 26 bits más significativos representan el número de marca (o del bloque al cual pertenece), mientras que los 6 bits menos significativos el desplazamiento a partir de la marca. Por ejemplo si se desea acceder a la línea 332 de la matriz de ocurrencia, los bits de k (o l) se interpretarían de la siguiente forma:

00000000000000000000000101 001100

Marca Desplazamiento

Es decir el valor deseado se encuentra en el bloque de la matriz de ocurrencia que inicia en la marca 5 y 12 posiciones delante de ésta. Basta ahora multiplicar este valor por el número de bytes que comprenden una línea del mapa conceptual (32) en este caso, de esta forma la dirección física que contiene los datos necesarios es 5 x 32 =160, o bien la dirección hexadecimal AOh.

4.4 Modificaciones adicionales al algoritmo

El algoritmo de alineación exacta presentado en el Listado 3.1, calcula en cada iteración nuevos valores de las variables k y l a partir de las ecuaciones:

$$k = C[\sigma] + Occ[\sigma, k-1] + 1 \quad [3]$$

$$l = C[\sigma] + Occ[\sigma, l] \quad [4]$$

Estas ecuaciones no representan ningún problema si se piensa en una implementación software. Sin embargo en una implementación hardware es deseable lograr estructuras lo más simétricas posibles. Para conseguir lo anterior, las ecuaciones 3 y 4 fueron reemplazadas en el algoritmo original por las siguientes:

$$k = C[\sigma] + Occ[\sigma, k] \quad [5]$$

$$l = C[\sigma] + Occ[\sigma, l] \quad [6]$$

Para no alterar los resultados, el vector de frecuencias ahora representará el número de caracteres en R que son lexicográficamente más pequeños que 'x' considerando el carácter especial $, y el arreglo de ocurrencias Occ contendrá el número de veces que el carácter ha aparecido en la TBW, hasta ese número de fila pero sin incluir ésta, además la condición de abandono prematuro de ciclo while debe cambiarse de $k \leq l$ *a* $k<l$.

4.5 Esquema general del sistema

Después de conocer las especificaciones del diseño, modificar apropiadamente el algoritmo para maximizar su simetría y analizar posibles estrategias para optimizar el uso de memoria y la velocidad de procesamiento, se establece en la Figura 4.4 el esquema general del sistema de aceleración, el cual consiste de tres módulos fundamentales: el productor, el procesador o firmware y el consumidor.

El consumidor es un módulo software programado en C y ejecutado en el procesador anfitrión, tiene como funciones el manejo de archivos, la creación de los índices de FM del genoma de referencia: Arreglo de sufijos, Transformada de Burrows-Wheeler, Matriz de Ocurrencia reducida y vector de frecuencia, el envío de los mismos y de las lecturas al módulo procesador, la detección y diagnóstico de la tarjeta de aceleración, y la configuración del dispositivo lógico programable mediante la carga del archivo con extensión .bit. Adicionalmente realiza una estimación de los resultados mediante funciones software.

El procesador es un módulo firmware y el principal del proyecto, consiste de múltiples bloques de alineación trabajando en paralelo, comunicados con el productor y el consumidor mediante el modelo de cadenas y con la memoria DRAM local mediante el protocolo AXI. Esta descrito en el lenguaje de descripción de hardware Verilog, y se ejecuta directamente en el FPGA de la tarjeta de aceleración.

Finalmente, el consumidor es otra función software programada en C y ejecutada en el procesador anfitrión, tiene como objetivo leer los resultados del módulo firmware, compararlos con la estimación software realizada por el productor y escribir los resultados con el formato apropiado en disco duro.

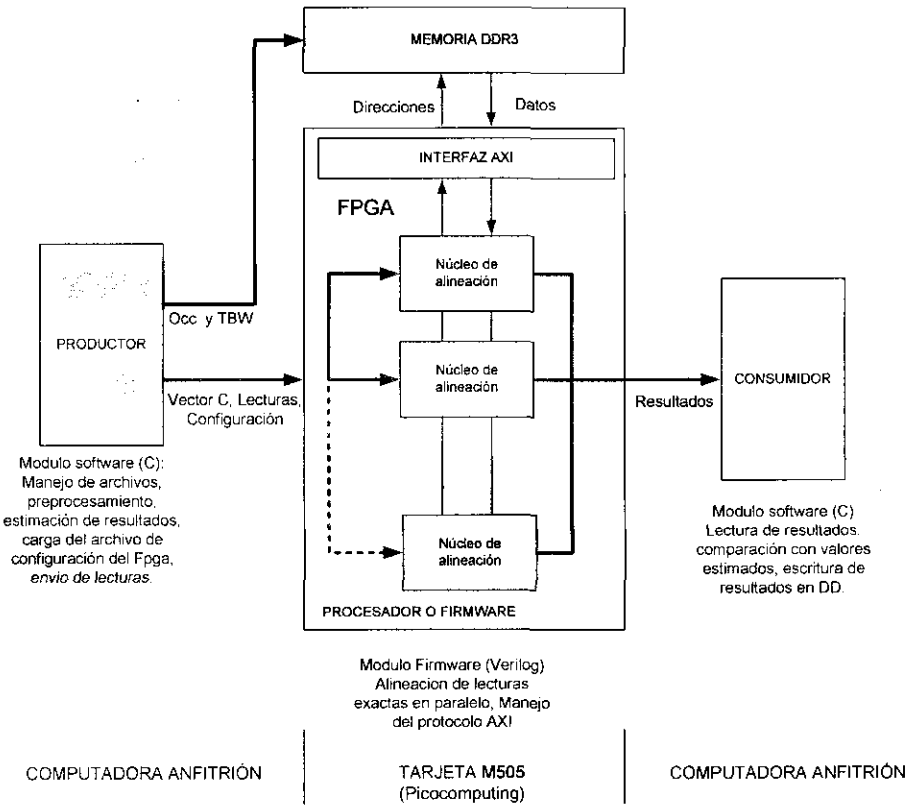


Figura 4.4 Diagrama general del sistema.

En conjunto los tres bloques trabajan de la siguiente manera:

1. El productor lee del disco duro el archivo de configuración del FPGA, verifica que exista una tarjeta de aceleración apropiada y configura el dispositivo lógico programable.
2. Obtiene del disco duro el genoma de referencia y genera sus índices de FM, incluyendo la matriz de ocurrencia reducida.
3. Lee una a una las lecturas y realiza una estimación de los resultados de la alineación.
4. Codifica cada carácter de TBW a dos bits y la envía junto con la matriz de ocurrencia reducida a la memoria DRAM de la tarjeta de aceleración.
5. Envía el vector de frecuencias y un comando de configuración a los registros del FPGA. Dicho comando contiene información de la longitud de las lecturas, el

número de estas, la posición del carácter especial ($) en la TBW y el tamaño del genoma de referencia.

6. Las lecturas codificadas previamente a dos bits se envían a los núcleos de alineación.
7. Cada núcleo en el firmware trabaja en forma independiente alineando una lectura, obteniendo la información de ocurrencias y la TBW de la memoria local de la tarjeta.
8. Cuando un núcleo termina su alineación comunica su estado al módulo consumidor.
9. El módulo consumidor recibe los resultados de la alineación (intervalo k y l), y los compara con los obtenidos mediante la estimación software.
10. El módulo consumidor informa al productor de un núcleo de alineación vacío y este carga una nueva lectura al mismo.
11. Cuando todas las lecturas se han procesado el consumidor almacena los resultados de la alineación en disco duro.

En la sección 4.7 se darán más detalles de los módulos productor y consumidor, en lo que sigue se describirá los bloques que constituyen al módulo productor o firmware, siendo este el tema central de la tesis.

4.6 Núcleo de alineación

El núcleo de alineación es el bloque principal del módulo firmware (Figura 4.5), en éste se implementa la función de alineación exacta presentada en el Listado 3.1, con las adaptaciones realizadas al principio de este capítulo. La unidad puede dividirse en 2 secciones: la trayectoria de datos y la unidad de control, la primera integrada a su vez por los registros de propósito específico, las unidades de procesamiento y el conmutador de memoria.

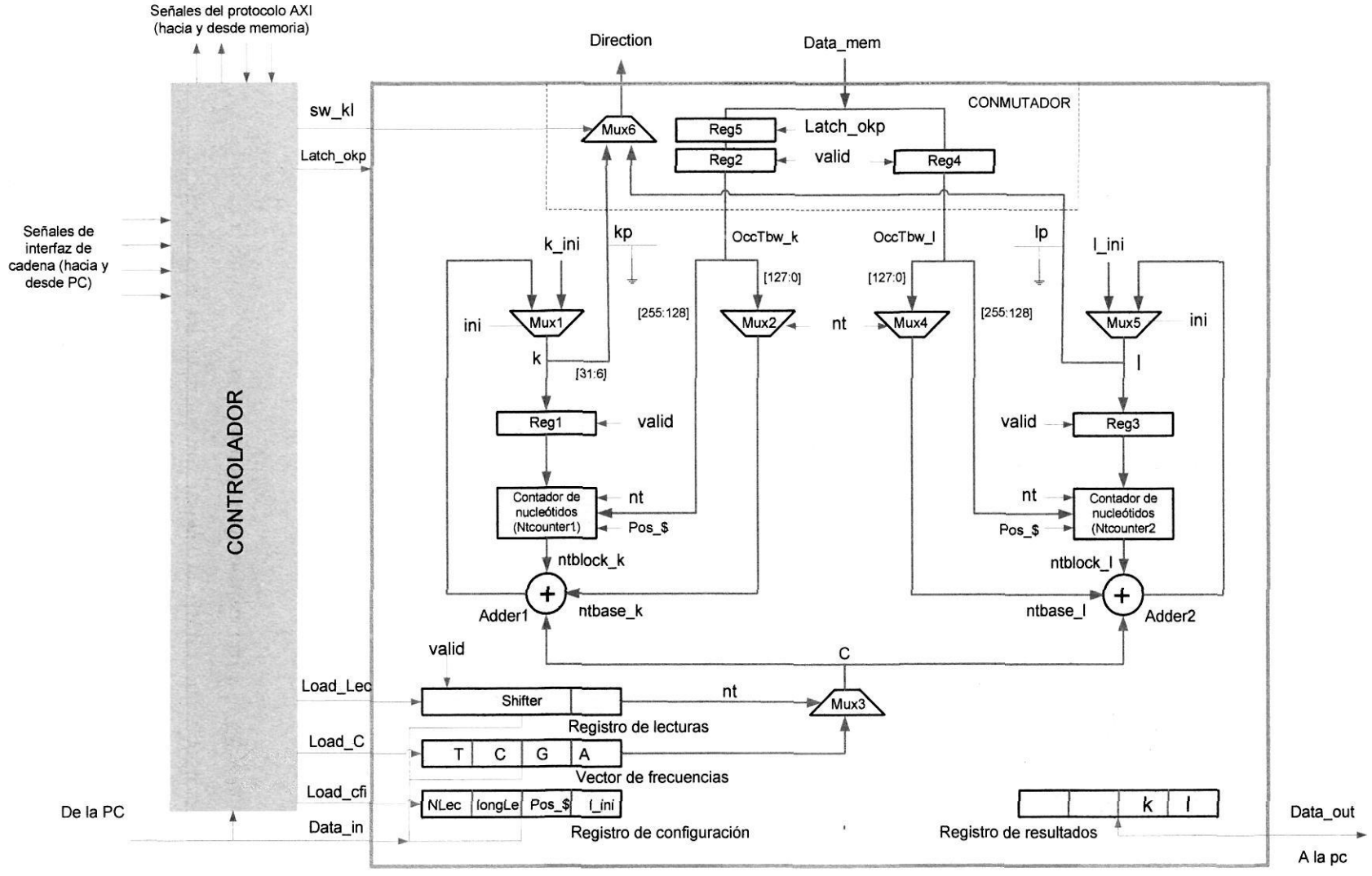


Figura 4.5 El núcleo de alineación exacta

4.6.1 Registros de propósito específico

Existen cuatro registros de propósito específico, cada uno con una longitud de 128 bits: el registro de configuración, el registro de lecturas, el vector de frecuencias y el registro de resultados.

El registro de configuración mantiene la información del proceso de alineación que se realiza actualmente y está organizado de la siguiente manera:

Bits 31:0	Longitud del genoma de referencia.
Bits 63:32	Posición del carácter especial ($) en la TBW.
Bits 95:64	Longitud de las lecturas.
Bits 127:96	Número de lecturas a procesar

El registro de lecturas, es un registro de desplazamiento, que contiene la lectura procesada actualmente. Debido a su longitud de 128 bits puede almacenar lecturas cortas de hasta 64 bits de longitud.

El registro vector de frecuencias, almacena cuatro enteros de 32 bits, cada uno representando a un elemento del vector del mismo nombre de acuerdo a la nomenclatura de los índices de FM.

Finalmente, el registro de resultados contiene los intervalos en donde la lectura alineada ha sido encontrada dentro de la referencia, parámetros k y l del algoritmo de alineación, y está organizada de la siguiente manera

Bits 31:0	Parámetro l.
Bits 63:32	Parámetro k.
Bits 95:64	No usado en la implementación actual.
Bits 127:96	No usado en la implementación actual.

4.6.2 Las unidades de procesamiento

Las unidades de procesamiento son los bloques que realmente procesan la información. La primera de ellas, opera para calcular el siguiente valor k del algoritmo, de acuerdo a la

ecuación 5 y está formado por los multiplexores Mux1 y Mux2, el registro Reg1, el contador de caracteres Ntcounter1 y el sumador Adder1. La segunda a su vez calcula el siguiente valor de *l*, de acuerdo a la ecuación 6 y está formado por los multiplexores Mux4 y Mux5, el registro Reg3, el contador de caracteres Ntcounter2 y el sumador Adder2. Ambas unidades tienen funcionalidad idéntica, por lo que en lo que sigue únicamente se describirá la primera de estas.

El multiplexor Mux1 elige si *k* contiene el valor inicial del algoritmo, es decir *k*=0 (o *l*=número de caracteres en la referencia, para Mux5 en la segunda unidad), o bien el resultado de la iteración previa. El valor de *k* se utiliza para calcular la dirección física de la información necesaria para hallar el valor de $O(\sigma,k)$, básicamente se extraen los bits 31:6 y se concatenan con 5 bits ceros a su derecha, con lo que se obtiene su multiplicación por 5, el resultado es la señal *kp* la cual es enviada a las líneas de dirección de memoria.

Los datos de lectura se reciben en la señal *OccTbw_k*, los 128 bits más significativos contienen el segmento de la TBW involucrado en el cálculo y se alimentan directamente a los contadores de nucleótidos, los 128 bits menos significativos contienen los cuatros valores de las marca y se alimenta al multiplexor Mux2, el cual transferirá a la señal *ntbase_k* uno solo de estos valores dependiendo del nucleótido procesado. En la parte inferior, el multiplexor Mux3 tiene como entrada los cuatro valores del vector de frecuencia y elige uno en su salida, conectada a la señal C, en función del nucleótido actual procesado.

Finalmente el contador de caracteres, es un módulo que dado un bloque de caracteres de la TBW, un nucleótido y una posición, entrega en su salida el número de veces que el nucleótido aparece desde el inicio del bloque y hasta la posición dada, su resultado se encuentra en la señal *ntblock_k*. Observe como la adición de la señal *ntbase_k* y *ntblock_k* representan el valor de $O(\sigma,k)$ de la ecuación 5 y si a este valor se suma el correspondiente a C, se obtiene el siguiente valor de *k*, esta es la función del sumador Adder1.

Inicialmente la señal *ini* es colocada a 0 por el controlador, con esto el multiplexor Mux1 coloca el valor inicial en *k*, a partir de este valor se integra en *kp* la dirección física de los datos involucrados en el cálculo y se envía a la memoria. Cuando el controlador de memoria tiene los datos de lectura disponibles, la señal *valid* es colocada a 1 habilitando al registro Reg1 quien transfiere la señal *k* al contador de caracteres. Al mismo tiempo la señal *valid* habilita al registro de desplazamiento (registro de lecturas), provocando el movimiento de su información, una posición a la derecha, la señal *nt* contiene ahora el código del primer nucleótido en la lectura, mismo que se transfiere al contador de caracteres y a los multiplexores Mux2 y Mux3, permitiéndole al primero hacer la estimación de la cantidad de nucleótidos del tipo *nt* en el segmento de la TBW desde el inicio y hasta la posición indicada por los bits 5:0 de *k*, y a los multiplexores transferir el dato correcto dentro de la marca y del vector de frecuencias respectivamente. El sumador realiza la adición de estos tres elementos, calculando el nuevo valor de *k*.

A partir de la siguiente iteración el valor de la señal *ini* permanecerá en 1, y la señal *k* simplemente tomará el valor calculado en la interacción previa. El ciclo se repite hasta que se procese el último carácter de la lectura o bien que el parámetro *k* sea menor al parámetro *l* calculado simultáneamente en la segunda unidad de procesamiento.

4.6.2.1 El contador de nucleótidos

Es el bloque más complejo de las unidades de procesamiento por lo que se le dedicará atención especial en este apartado. Como se indicó, su función consiste en calcular el número de nucleótidos del tipo específico que existen en un segmento de la TBW, desde su inicio y hasta una posición determinada. El problema se complica al recordar que el carácter especial $ fue codificado como una 't' dentro de la TBW, así que debe existir un medio para distinguir esa 't' y entregar el valor correcto a la salida. Adicionalmente se desea que el modulo cuente hasta 64 nucleótidos de la cadena de una forma muy rápida por lo que se requiere una estructura altamente paralela y con la mínima longitud horizontal posible.

La estructura del módulo propuesto se muestra en la Figura 4.6. En esta figura, *tbw* es una señal de 128 bits que contiene el segmento de la TBW sobre la que se llevará el conteo. La señal de dos bits *nt* representa el tipo de nucleótido que se desea contar. *direction* es una señal de 32 bits que contiene en sus bits 5:0 la posición hasta donde debe contarse, mientras que el resto de los bits (31:6) juntos con la señal de 32 bits *primary*, la cual contiene el valor de la posición del carácter especial $ en la TBW, se utilizan en la etapa de corrección al final del diseño.

Inicialmente, la señal *tbw* se convierte, mediante el filtro nt, de su representación original en 128 bits a una representación de 64 bits en función del nucleótido en cuestión, si en alguna posición ocurre el nucleótido de interés, el bit correspondiente se coloca a 1 o a 0 en caso contrario. La nueva representación es operada mediante el bloque AND con una máscara de unos hasta la posición de interés, eliminando los bits que no deben ser contados. De aquí en adelante el problema se reduce a contar la población de los bits restantes. Para esto el diseño implementa codificadores de 8:4 y un árbol de 7 sumadores para llegar al resultado parcial *sum7*. Cada codificador opera sobre un segmento de 8 bits y en su salida representa el código correspondiente al número de bits iguales a 1 en su entrada.

El carácter especial fue codificado por el software como una 't' para poder codificar la TBW con solo dos bits por carácter. Si esta 't' pertenece al mismo bloque de la matriz de ocurrencia en el que reside '$' y si el desplazamiento de 't' en ese bloque es mayor al correspondiente de '$', es evidente que se estará contando una Timina de más equivocadamente, por lo que debe restarse una unidad al resultado parcial *sum7*. Esta función la realizan en conjunto el corrector y el restador en la parte inferior del diseño. Es decir, considerando que cada dirección en el sistema puede interpretarse como integrada por dos secciones: bloque o marca (bits 31:6) y desplazamiento (bits 5:0), entonces el corrector simplemente implementa la condición:

```
if ( nt == 't' && direction[31:6] == primary[31:6] && direction[5:0] > primary[5:0] ) {
corrector_out=1};
```

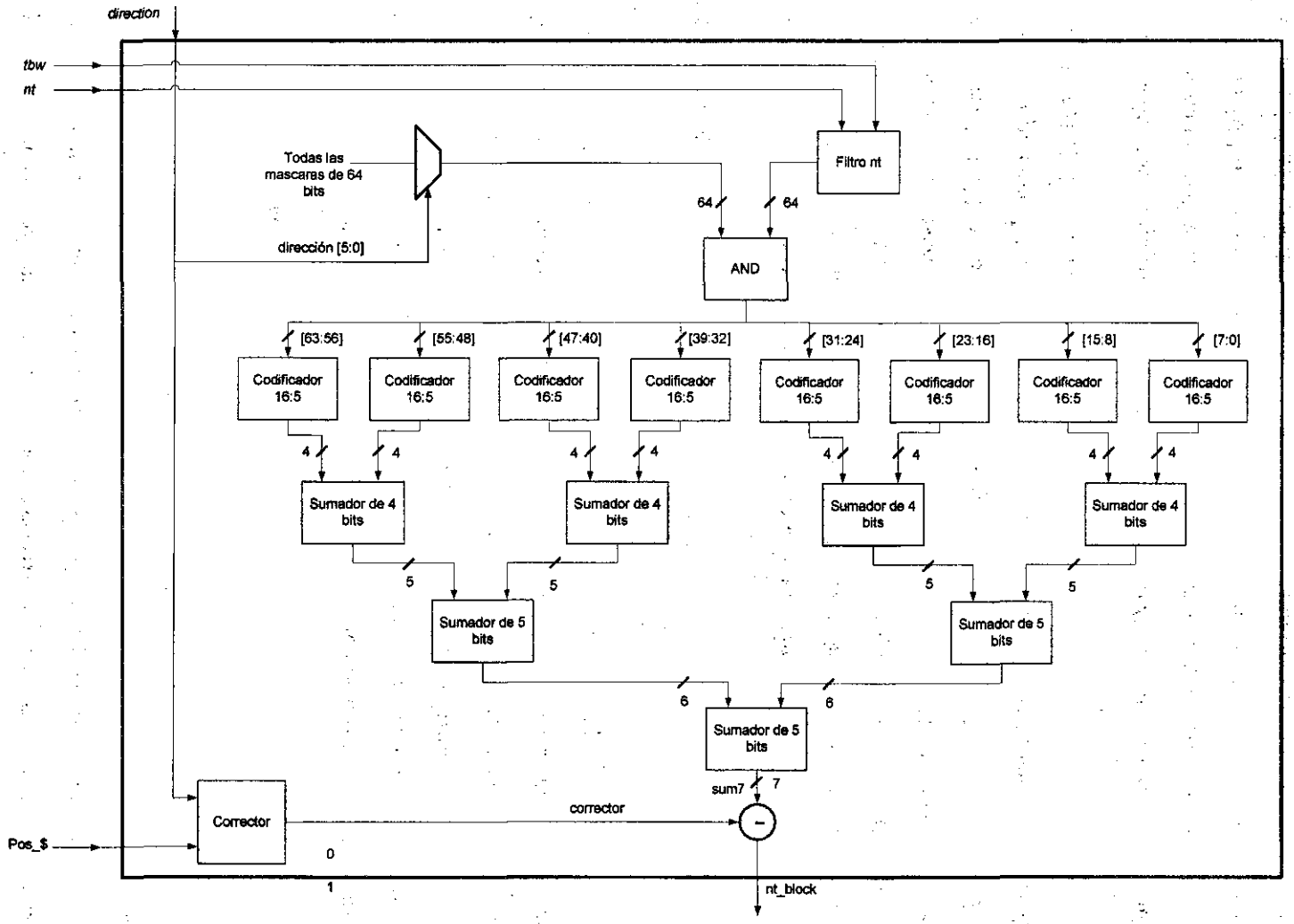


Figura 4.6 El contador de nucleótidos.

4.6.3 El conmutador de memoria

Esta constituido únicamente por los registros Reg2, Reg4 y Reg 5 y el multiplexor Mux6 y tiene como función multiplexar los accesos a memoria de las unidades de procesamiento, para utilizar solo un controlador de memoria en la tarjeta de desarrollo. Este módulo podría llegar a ser innecesario si se utilizará una tarjeta con múltiples controladores, duplicando de esta forma la velocidad de procesamiento.

4.6.4 El controlador

El controlador es una máquina de estados tipo Mealy que no solo dirige el flujo de datos en el resto de los bloques que integran el núcleo de alineación, sino que, además genera las señales apropiadas para comunicarse con el procesador anfitrión y con el sistema de memoria, por esta razón se pospone su explicación a la siguiente sección una vez que se haya descrito la forma en que la tarjeta de desarrollo interactúa con el procesador (mediante cadenas) y con el subsistema de memoria (mediante el protocolo AXI).

4.7 Implementación.

En este proyecto solo fue implementado un núcleo de alineación, con esto se verifica la funcionalidad del mismo y puede estimarse el área y la velocidad de un sistema de múltiples núcleos. La plataforma en la cual fue implementado el diseño es una computadora de escritorio con procesador Intel Core i5 4460 de cuarta generación, memoria DRAM de 8 GB, con puertos de expansión PCIe 2 x 16, versión 2.0 y el sistema operativo Ubuntu 14.04. A este equipo se le integró la tarjeta de desarrollo M-505-k325T de la empresa Pico Computing.

El diseño fue simulado y depurado utilizando SWsim, un sistema de simulación manejada por software proporcionado por Pico Computing, en conjunto con Modelsim versión 10.1.d, y sintetizado e implementado mediante el entorno de desarrollo Vivado versión 2014.2. Los detalles de la plataforma así como del proceso de implementación se dan en la presente sección.

4.7.1 La tarjeta de aceleración M-505-k325T

La tarjeta M-505-k325T de la empresa Pico Computing es un sistema de cómputo de alto desarrollo basada en el bus PCI estándar, diseñado para maximizar la memoria y el ancho de banda lógico. Su núcleo central es un FPGA de la familia Kintex-7, el K325T u opcionalmente el k410T, incluye también un subsistema de memoria local DDR SODIMM de 8 GB con controlador de memoria independiente y una estructura de comunicación PCIe x8 Gen 2 completamente conmutado para establecer la comunicación con la computadora anfitrión y el sistema de memoria local. La Figura 4.7 muestra la imagen frontal de la tarjeta junta al resumen de sus características.

La tarjeta M-505-k325t se conecta a la computadora anfitrión mediante el módulo M-500 de la misma empresa sirviendo como *backplane.* Este último puede albergar hasta 6 tarjetas M-505-k325t logrando una elevada densidad de procesamiento paralelo para aplicaciones de cómputo intensivo.

- FPGA Kintex-7 XC7K325T
- Interface PCI expreses Gen 2 x 8
- Hasta 6 módulos M-505 por blackplane EX500
- 8 GB DDR3 SODIMM
- 128 MB NOR Flash
- 34 LVDS
- 8 GTX Transceptores

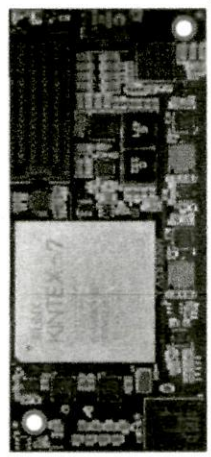


Figura 4.7 Características de la tarjeta de desarrollo M505-k325t

Pico Computing proporciona una Interfaz de Programación de Aplicaciones (API) para las plataformas LINUX, la cual provee el enlace entre el software de aplicación corriendo en la computadora anfitrión y el algoritmo hardware, o firmware, implementado en el FPGA. La API incluye las librerías software y sus correspondientes

componentes firmware que permiten el control del FPGA y el procesamiento, la meta de esta interface es abstraer los detalles de comunicación y proveer un método potente y relativamente fácil de programar el sistema.

La API se distribuye como un conjunto de archivos fuentes incluidos en el paquete de instalación Pico. El punto de entrada principal es la clase PicoDrv, la cual representa un FPGA. Si por ejemplo se tienen 3 módulos M-505 en el sistema, deben crearse 3 objetos PicoDrv, cada uno correspondiendo a uno de los FPGAs.

La Pico API define dos modelos para comunicar el firmware con el software, el primero es el tradicional basado en bus, mientras que el otro es el modelo de cadenas de datos.

4.7.1.1 Modelo basado en bus (PicoBus)

El modelo basado en bus es el paradigma mapeado a memoria de la mayoría de los dispositivos digitales. En este modelo el firmware se considera como un esclavo sobre un bus mapeado a memoria y se controla por software corriendo en el CPU anfitrión. El PicoBus puede usarse tanto para escribir datos al firmware desde el software como para leer datos del firmware al software, en cada transacción, el modelo involucra el manejo de un dato y una dirección, y está intentado para casos donde solo se necesita mover pocos megabytes de datos por segundo.

4.7.1.2 Modelo basado en cadenas (PicoStreams)

El modelo basado en cadenas está basado en el concepto de comunicación secuencial de procesos (CSP), en este, una cadena es considerada una secuencia de datos con control de flujo. La API permite conocer cuando la cadena está disponible para leer o escribir datos y el usuario puede decirle a la API cuando está listo para enviar o recibir datos. Las cadenas son una abstracción simple del modelo basado en bus, en estas no hay direcciones, los datos simplemente se envían o reciben en forma ordenada. Esta simplicidad permite que las cadenas sean más eficientes para mover cantidades grandes de datos en un modo rápido y eficiente.

Una cadena es un canal uni-direccional de comunicación, de esta forma hay dos tipos de cadenas, las de entrada al FPGA y las de salida desde el FPGA. Típicamente un sistema incluye una cadena en cada dirección estableciendo de esta forma una comunicación bidireccional.

Las cadenas no solo limitan su uso a la comunicación entre el CPU anfitrión y el FPGA, sino que pueden usarse para comunicar directamente FPGAs sin la intervención del anfitrión, logrando de esta forma sistemas de muy baja latencia.

Debido a sus ventajas y simplicidad el modelo de cadenas fue elegido para el desarrollo de este proyecto de tesis.

4.7.1.3 Acceso a memoria

La interfaz a memoria en la tarjeta M505 se realiza mediante el protocolo AXI (del inglés Advanced eXtensible Interface). AXI es un protocolo estándar que utiliza 5 canales de comunicación independientes:

- Canal de direcciones de escritura
- Canal de datos de escritura
- Canal de respuesta a la escritura
- Canal de direcciones de lectura
- Canal de datos de lectura

El protocolo AXI define las interacciones entre un maestro y un esclavo en esos 5 canales. Una operación de escritura utiliza los canales de datos de escritura y direcciones de escritura, además del canal de respuesta a la escritura por donde se recibe una confirmación por cada ráfaga escrita (ver Figura 4.8). Las operaciones de lectura se inician en el canal de direcciones de lectura y los datos retornan por el canal de datos de lectura (ver Figura 4.9). Todos los canales son completamente independientes, así, las operaciones de lectura y escritura pueden realizarse simultáneamente.

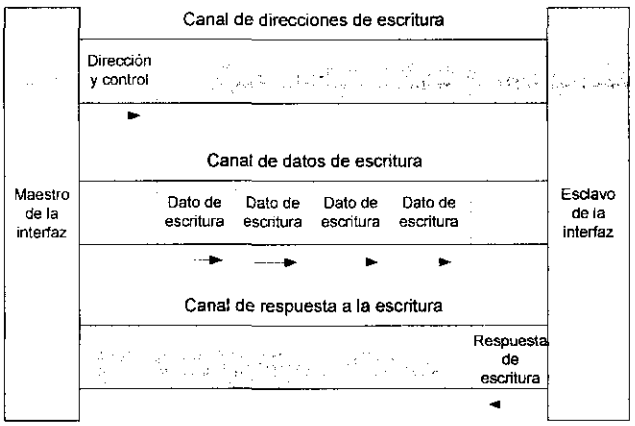


Figura 4.8 Canales del protocolo AXI relacionados con la escritura.

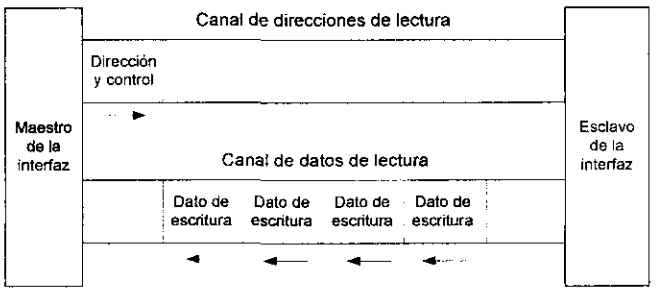


Figura 4.9 Canales del protocolo AXI relacionados con la lectura.

En el protocolo AXI las direcciones son especificadas por byte, es decir es direccionable por byte, además, los datos pueden entregarse en ráfagas de hasta 256 transferencias por ciclo requiriendo una sola dirección. Para clarificar, la lógica de usuario puede enviar una dirección (en los canales de direcciones de lectura o de direcciones de escritura) por 256 transferencias de datos (en los canales de datos de lectura o datos de escritura).

Cada canal en el protocolo AXI utiliza un simple *handshake valid-ready* donde los datos solo se mueven desde el maestro al esclavo o del esclavo al maestro únicamente si las señales *valid* y *ready* son puestas a uno simultáneamente. De esta forma, cada uno de los 5 canales tienen sus propias señales *handshake valid-ready.*

Una descripción completa de las señales del protocolo AXI puede encontrarse en http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022d/index.html.

4.7.1.4 Marco de trabajo Firmware (Pico Framework)

Pico Computing provee además un marco de trabajo Firmware escrito en Verilog, denominado Pico Framework el cual provee el acceso a toda la funcionalidad del FPGA. Sus componentes fundamentales son un controlador PCIe DMA y una interface de memoria multipuerto. Al construir un archivo de configuración para el FPGA el Pico Framework será siempre el nivel superior y el módulo del usuario se instanciará dentro del mismo.

4.7.2 Firmware del proyecto

Este proyecto usa dos cadenas, una de entrada y otra de salida, además de la interface a memoria de la tarjeta. La forma de especificar esto al Pico Framework es mediante el archivo PicoDefine.v. El Pico Framework toma cuidado de implementar el soporte lógico para conectar esas cadenas a la interface PCIe, lo cual dará el acceso a la computadora anfitrión. El contenido de este archivo es el que se muestra en el **¡Error! No se encuentra el origen de la referencia.**.

Listado 4.1 El archivo de configuración PicoDefine.v

```
//Definición del nombre del módulo de usuario
`define USER_MODULE_NAME Aligner_v1

//Definición del controlador DDR
`define PICO_DDR3
`define PICO_1_AXI_MASTERS

//Definición de las cadenas de entrada y de salida
`define STREAM1_IN_WIDTH 128
`define STREAM1_OUT_WIDTH 128

// Definición del FPGA la tarjeta utilizada.
`define PICO_MODEL_M505
`define PICO_FPGA_LX325T
```

La primera línea de código define el nombre del módulo de usuario, Aligner_v1, este módulo contendrá la lógica del núcleo de alineación discutido en la sección 4.6.

Posteriormente se especifica el uso de la interface DDR3 definiendo PICO_DDR3 y en la siguiente línea el número de puertos a utilizar en el diseño, en este caso 1 puesto que se desea utilizar simultáneamente los 256 bits disponibles en la interface. Las dos líneas siguientes definen las cadenas de entrada y de salida al diseño, ambas de 128 bits, esto es conveniente puesto que los datos enviados a registros del FPGA todos son de esta magnitud (vector C, palabra de configuración y lecturas). Finalmente se define el modelo de la tarjeta y el FPGA específico que se utilizará.

El Pico Framework ve estas definiciones y conecta las dos cadenas al módulo. La lista de puertos relacionados con las cadenas en el módulo Aligner_v1 se observa en el Listado 4.2. Cuando el software de la aplicación envía un dato, lo indica poniendo *s1i_valid* en alto, el firmware puede responder mediante la señal *s1i_rdy*, con esto el dato en *s1o_data* queda grabado en alguno de los registros del núcleo de alineación. Al terminar la alineación de alguna lectura los resultados se envían colocándolos en *s1o_data* a través del registro de resultados, y poniendo la señal *s1o_valid* en alto, el software responderá con *s1o_rdy* en el momento en que acepte los datos. Las señales *clk* y *rst*, representan las señales de reloj y reset comunes a las cadenas.

Listado 4.2 Puertos relacionados con cadenas en el módulo Aligner_v1

```
module Aligner_v1(

//Señales comunes a las cadenas
input           clk,
input           rst,

//Cadena de entrada
input           s1i_valid,
input [127:0]   s1i_data,
output          s1i_rdy,

//Cadena de salida
output reg      s1o_valid,
output [127:0]  s1o_data,
input           s1o_rdy,
```

Adicionalmente, el Pico Framework creará la interfaz a memoria con todas las señales del protocolo AXI descrito previamente. La lista de puertos relacionados con la interfaz a memoria del módulo Aligner_v1 se observa en el Listado 4.3.

Listado 4.3 Puertos del protocolo AXI para interface a memoria en el módulo Aligner_v1

```
// Señales AXI del canal de direcciones de escritura

input                 c0_s1_axi_awready,    // Señal listo del handshake
output [7:0]          c0_s1_axi_awid,       // Identificador de escritura
output reg [32:0]     c0_s1_axi_awaddr,     // Dirección de escritura
output reg [7:0]      c0_s1_axi_awlen,      // Longitud de ráfaga de escritura
output [2:0]          c0_s1_axi_awsize,     // Tamaño de ráfaga de escritura
output [1:0]          c0_s1_axi_awburst,    // Tipo de ráfaga de escritura
output                c0_s1_axi_awlock,     // Candado de escritura
output [3:0]          c0_s1_axi_awcache,    // Configuración de cache
output [2:0]          c0_s1_axi_awprot,     // Tipo de protección de escritura
output [3:0]          c0_s1_axi_awqos,      // Señalización QoS de escritura
output                c0_s1_axi_awvalid,    // Señal valido del Handshake


// Señales AXI del canal de datos de escritura

input                 c0_s1_axi_wready,     // Señal listo del handshake
output [255:0]        c0_s1_axi_wdata,      // Dato de escritura
output [30:0]         c0_s1_axi_wstrb,      // Señal strobes de escritura
output                c0_s1_axi_wlast,      // Ultima transacción de escritura
output                c0_s1_axi_wvalid,     // Señal valido del handshake


// Señales AXI del canal de respuesta a la escritura

input  [7:0]          c0_s1_axi_bid,        // ID de respuesta
input  [1:0]          c0_s1_axi_bresp,      // Respuesta de escritura
input                 c0_s1_axi_bvalid,     // Señal valido del handshake
output                c0_s1_axi_bready,     // Señal listo del handshake


// Señales AXI del canal de direcciones de lectura

input                 c0_s1_axi_arready,    // Señal listo del handshake
output [7:0]          c0_s1_axi_arid,       // Identificador de lectura
output [32:0]         c0_s1_axi_araddr,     // Dirección de lectura
output [7:0]          c0_s1_axi_arlen,      // Longitud de ráfaga de lectura
output [2:0]          c0_s1_axi_arsize,     // Tamaño de ráfaga de lectura
output [1:0]          c0_s1_axi_arburst,    // Tipo de ráfaga de lectura
output                c0_s1_axi_arlock,     // Candado de lectura
output [3:0]          c0_s1_axi_arcache,    // Configuración de cache
output [2:0]          c0_s1_axi_arprot,     // Tipo de protección de lectura
output reg            c0_s1_axi_arvalid,    // Señal valido del handshake
output [3:0]          c0_s1_axi_arqos,      // Señalización QoS de lectura


// Señales AXI del canal de datos de lectura

input  [7:0]          c0_s1_axi_rid,        // Identificador de lectura
input  [1:0]          c0_s1_axi_rresp,      // Respuesta de lectura
input                 c0_s1_axi_rvalid,     // Señal valido del handshake
input  [255:0]        c0_s1_axi_rdata,      // Dato de lectura
input                 c0_s1_axi_rlast,      // Última transacción de lectura
output                c0_s1_axi_rready      // Señal listo del handshake
);
```

Las señales de entrada provienen de la interfaz AXI, mientras que las de salida deben de ser asignadas por la lógica de usuario. En esta aplicación en particular no existen operaciones de escritura por lo que lo que las señales AXI de los canales de direcciones de escritura, datos de escritura y respuesta de escritura, no se utilizaron.

Por su parte las señales del canal de direcciones de lectura y datos de lectura se clasificaron en dos grupos, el primero comprende las señales cuyo valor es constante durante la operación del acelerador, y el segundo aquellas cuyo comportamiento es dinámico, es decir varían durante la ejecución. El primer grupo de señales junto con su asignación constante se muestra en el Listado 4.4.

Listado 4.4 Grupo de señales AXI constantes en el diseño.

```
//Señales del puerto de datos de escritura

assign c0_s1_axi_wstrb    = ~0;                  // Escritura por Byte habilitada


//Señales del puerto de direcciones de lectura

assign c0_s1_axi_arlock    = `NORMAL_ACCESS;         //Acceso normal
assign c0_s1_axi_arcache   = `NON_CACHE_NON_BUFFER; //Sin buffer, ni cache
assign c0_s1_axi_arprot    = `DATA_SECURE_NORMAL;    //Seguridad de datos normal
assign c0_s1_axi_arburst   = `INCREMENTING;          //Incrementando
assign c0_s1_axi_arqos     = `NOT_QOS_PARTICIPANT;   //Sin participar en QoS
assign c0_s1_axi_arsize    = `THIRTY_TWO_BYTES;      //32 bytes por transferencia
assign c0_s1_axi_arid      = 0;                      //No se requiere identificador
assign c0_s1_axi_arlen     = 0;                      //longitud en términos de
                                                     //transferencias de 256 bits,
```

El segundo grupo, las señales dinámicas juegan el papel más importante, el éxito en un acceso de lectura a memoria depende de su correcta sincronización, y son manipuladas por la máquina de estados de la unidad de control. En lo que sigue únicamente se indicará su funcionalidad.

Señales del puerto de direcciones de escritura

c0_s1_axi_araddr	Dirección de escritura definida en bytes.
c0_s1_axi_arvalid	Esta señal valida la dirección de escritura, debe colocarse en alto cuando dicha dirección se encuentre lista.

c0_s1_axi_arready	El controlador de memoria habilita esta señal si esta lista para aceptar nuevas direcciones de memoria. La dirección de lectura se acepta si c0_s1_axi_arvalid y c0_s1_axi_arready están en alto en el mismo ciclo de reloj.

Señales del puerto de datos de lectura

c0_s1_axi_rdata	Datos de lectura provenientes de la memoria RAM
c0_s1_axi_rvalid	Esta señal valida los datos de lectura, es puesta en alto por el controlador de memoria cuando ha colocado datos validos en la línea correspondiente
c0_s1_axi_rready	La lógica de usuario debe habilitar esta señal si esta lista para recibir nuevos datos de lectura. Los datos de lectura son grabados si c0_s1_axi_rvalid y c0_s1_axi_rready son colocados en alto en el mismo ciclo de reloj.

4.7.2.1 Conexión del núcleo de alineación al Pico Framework

La trayectoria de datos del núcleo de alineación, fue descrita en Verilog en una forma completamente estructural, de esta manera se asegura su correcta interpretación por la herramienta de síntesis incorporado en el entorno de desarrollo Vivado.

Para poder operar apropiadamente con el procesador anfitrión y con la memoria DDR en la tarjeta de desarrollo, las señales externas de la trayectoria de datos fueron conectadas con las correspondientes en el protocolo AXI y en las cadenas de entrada y de salida, del Pico Framework. Específicamente la señal *dir* conteniendo la dirección física para hallar los datos y calcular un valor de ocurrencia fue conectada con c0_s1_axi_araddr. La señal *OccBwt*, la cual es en donde se espera recibir los datos de memoria, se conectó a la señal c0_s1_axi_ardata. Finalmente, las señales de entrada y salida de datos *data_in* y *data_out* se conectaron a las señales de cadena *s1i_data* y *s1o_data* respectivamente. Las correspondientes asignaciones se muestran en el Listado 4.5:

Listado 4.5 Asignaciones para conectar la trayectoria de datos al Pico Framework.

```
assign slo_data = data_out;
assign data_in  = sli_data;
assign c0_s1_axi_araddr = dir;
assign OccBwt = c0_s1_axi_ardata;
```

Por su parte, la unidad de control dirige el flujo de la trayectoria de datos y establece la comunicación con el exterior a través del manejo de las señales dinámicas del protocolo AXI y de las señales de cadena. Esta función se implementa mediante la máquina de estados de la Figura 4.10 y dos contadores auxiliares. El primer contador lleva la cuenta del número de lecturas procesadas (Rcounter), mientras que el segundo hace lo propio con el número de nucleótidos procesados (Ntcounter).

La máquina permanece en el estado INICIO hasta que el software le envía el primer dato que corresponde a los cuatro enteros del vector C, esto lo indica con la señal *s1i_valid*, al hacerlo la señal *load_vector* se pone en alto habilitando al registro vector C y grabando en este último la información, posteriormente pasa al estado CONFIGURACIÓN.

En los estados CONFIGURACIÓN y CARGA_LECTURAS, ocurren operaciones similares cargando los registros de configuración y de lecturas, al habilitarlos mediante las señales *load_confi* y *load_read*. También se resetean lo contadores Rcounter y Ntcounter para comenzar el conteo de las lecturas y nucleótidos respectivamente.

Los estados de direccionamiento envían a memoria las direcciones físicas de los datos para el cálculo de los valores de ocurrencia. Si se procesa el primer nucleótido, la señal *ini* es mantenida en bajo en ambos estados, transfiriendo los valores iniciales al sistema. Si esto no ocurre la señal *ini* se coloca a alto para permitir pasar a *k* y *l* los valores de la iteración anterior. En el estado DIRECCIONA_DDR1 la señal *sw_kl* se encuentra en bajo con lo cual se asegura enviar la dirección física *kp*, y en el estado DIRECCIONA_DDR2, se pone en alto para enviar la dirección física *kl*. Cada uno de estos estados pasa al siguiente cuando el controlador de memoria acepta las direcciones colocando en alto la señal c0_s1_axi_arready.

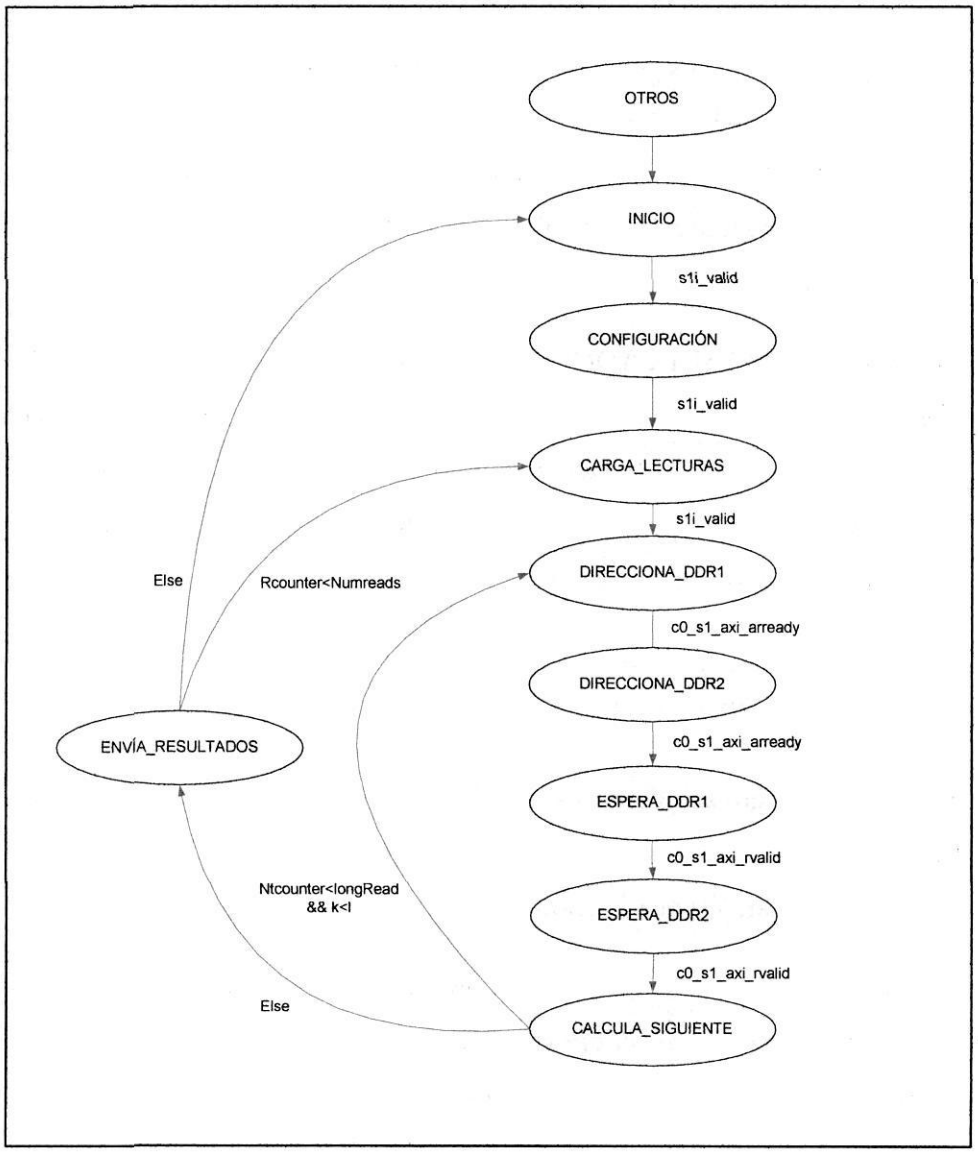


Figura 4.10 Máquina de estados que implementa la unidad de control del núcleo de alineación.

En los estados ESPERA_DDR1 y ESPERA_DDR2, la máquina espera los datos de lectura que corresponden a las direcciones en *kp* y *lp* respectivamente. Cuando estos llegan, el controlador de memoria lo indica poniendo en alto la señal c0_s1_axi_arvalid, lo que permite a la máquina pasar a los siguientes estados. Adicionalmente al llegar el resultado de la segunda lectura la señal *valid* se coloca en alto, permitiendo el paso de los datos recién recibidos y actualizando todos los valores en el núcleo de alineación.

El estado CALCULA_SIGUIENTE proporciona el tiempo necesario para que las unidades de procesamiento calculen los nuevos valores de *k* y de *l*, si aún no se han procesados todos los nucleótidos de la lectura y el parámetro *k* es menor a *l*, la máquina regresará al estado DIRECCIONA_DDR1 para iniciar el procesamiento del siguiente nucleótido. En caso contrario se pasa al estado ENVIA_RESULTADOS para enviar al software los resultados de alineación.

En el estado ENVIA_RESULTADOS la máquina pone en alto la señal s1o_valid con lo cual informa al software que tiene listo el resultado, si este está disponible para aceptarlo responde habilitando la señal s1o_rdy, logrando su transferencia. La máquina entonces regresa al estado CARGA_LECTURA si existen más lecturas por procesar o al estado INICIO en caso contrario.

4.7.3 Software del proyecto

El software del proyecto se programó en C basado en el diagrama de flujo de la Figura 4.11. En lo que sigue no se pretende una descripción detallada de la programación del mismo, solo se hará referencia a funciones estratégicas para el manejo de la tarjeta.

La identificación de la tarjeta y la carga del archivo con extensión .bit en el FPGA, se realiza mediante la función *RunBitFile* tomando el nombre del archivo desde la línea de comandos, como se muestra en el Listado 4.6:

Listado 4.6 Identificación de la tarjeta y carga del archivo .bit en el FPGA.

```
if (argc < 4) {
        fprintf(stderr, "Debe especificar el archivo .bit, archivo de
        referencias y archivo de lecturas.\n");
        exit(1);
}else{
        bitFileName = argv[1];

// La función RunBitFile localiza una tarjeta que pueda ejecutar
// el archivo con extensión .bit
printf("\nCargando el FPGA con el archivo '%s' ...\n", bitFileName);
err = RunBitFile(bitFileName, &pico);
if (err < 0) {
  // La función PicoErrors_FullError decifra los codigos de error de RunBitFile.
  fprintf(stderr, "RunBitFile error: %s\n", PicoErrors_FullError(err, ibuf,
  sizeof(ibuf)));
  exit(1);
  }
}
```

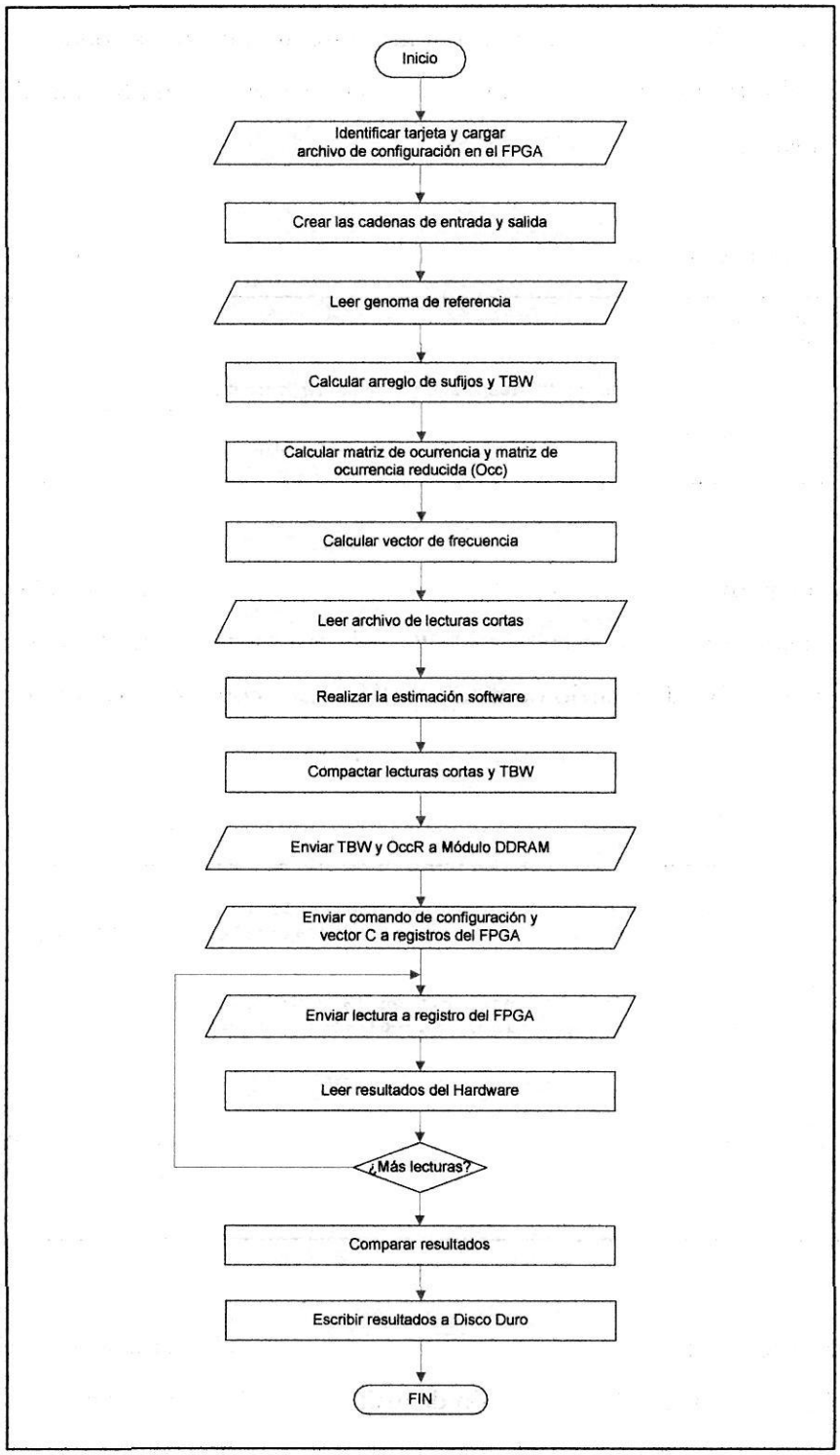


Figura 4.11 Diagrama de flujo del software del proyecto.

El resultado es un *handle* con el cual se abre la cadena para establecer la comunicación con el FPGA mediante la función *CreateStream*, esto último se muestra en el Listado 4.7. Es importante resaltar que, a diferencia del firmware, únicamente se define un nombre de cadena y el software creará (y manipulará) con este, una de entrada y una de salida en forma automática.

Listado 4.7 Apertura de la cadena para comunicación con el FPGA

```
// Creación de la cadena 1
stream = pico->CreateStream(1);
if (stream < 0) {
                fprintf(stderr, "Incapaz de crear la cadena 1: %s\n",
                                PicoErrors_FullError(stream, ibuf, sizeof(ibuf)));
                 exit(1);
                 }
```

Una vez obtenidos los índices de FM y realizada la estimación software, el envió de la TBW y la matriz OccR a la memoria DDR de la tarjeta de desarrollo se lleva a cabo utilizando la función de manejo de memoria *WriteRam*, como se observa en el Listado 4.8.

Listado 4.8 Envió de la TBW y la matriz OccR a DDR.

```
printf("\nEnviando la matriz de ocurrencia comprimida y la TBW a DDR3\n");
        err = pico->WriteRam(0, index, (n/blocksize+1)*8*4, PICO_DDR3_0);

    if (err < 0){
        fprintf(stderr, "Error al escribir en la RAM: %s\n",
        PicoErrors_FullError(err, ibuf, sizeof(ibuf)));
        exit(1);
        }

   else if (err != (n/blocksize+1)*8*4){
        fprintf(stderr, "WriteRam escribiÃ³ %i bytes en lugar de los deseados 16
        bytes\n",err);
        exit(1);
    }
```

Por su parte, el envió del comando de configuración y el vector C se realiza en el Listado 4.9, para esto se utiliza la función de manejo de cadenas *WriteStream*.

Listado 4.9 Envió de del vector de frecuencias y el comando de configuración al FPGA.

```
err = pico->WriteStream(stream, C, 16);
if (err < 0){
     fprintf(stderr, "WriteStream error: %s\n", PicoErrors_FullError(err, ibuf,
     sizeof(ibuf)));
      exit(1);
      }
else if (err != 16) {
      fprintf(stderr, "write error. returned %i, but should have been %i\n", err,
      16);
      exit(1);
              }
err = pico->WriteStream(stream, cmd, 16);
    if (err < 0){
        fprintf(stderr, "WriteStream error: %s\n", PicoErrors_FullError(err, ibuf,
        sizeof(ibuf)));
       exit(1);
              }
    else if (err != 16) {
        fprintf(stderr, "write error. returned %i, but should have been %i\n",
        err, 16);
         exit(1);
              }
```

Finalmente, el envío de la lectura y recepción dc resultados utilizan las funciones de manejo de cadena *WriteStream* y *ReadStream* respectivamente. Esto se muestra en el Listado 4.10.

Listado 4.10 Envío de lecturas y recepción de resultados

```
for (q=0; q<numreads;q++){
       for (j=0;j<4;j++){
              readcurrent[j]=read_bin[4*q+j];

              err = pico->WriteStream(stream, readcurrent, 16);
              if (err < 0){
                     fprintf(stderr, "WriteStream error: %s\n",
                     PicoErrors_FullError(err, ibuf, sizeof(ibuf)));
                     exit(1);
                     }
              else if (err != 16) {
                     fprintf(stderr, "write error. returned %i, but should have
                     been %i\n", err, 16);
                     exit(1);
                     }
              //Esperamos resultados
              if ((err = pico->ReadStream(stream, resHardware, 16)) < 0){
              fprintf(stderr, "ReadStream error: %s\n",
              PicoErrors_FullError(err, ibuf, sizeof(ibuf)));
              exit(1);
              }
```

Una descripción detallada de las funciones de la Pico API, pueden hallarse en la documentación de la tarjeta, disponible en: https://picocomputing.zendesk.com/home.

5 RESULTADOS

En este capítulo se presentan los principales resultados del diseño. La primera sección comprende la verificación funcional del mismo, para esto se muestra detalladamente la ejecución del acelerador alineando una sola lectura, mostrando la salida paso a paso así como las formas de onda en el interior del FPGA. La segunda sección caracteriza la razón de procesamiento del sistema, realizando múltiples pruebas, en las cuales diferentes conjuntos de lecturas generadas artificialmente mediante el programa Wgsim se alinean a diferentes cromosomas humanos, midiendo el tiempo de procesamiento y comparándolo con su contraparte software. La siguiente sección analiza el uso de área en el chip y el consumo de potencia. Finalmente la última sección estima los resultados esperados para una implementación con múltiples núcleos de alineación y los compara con trabajos relacionados.

5.1 Funcionalidad del sistema

Para validar y poder mostrar la correcta funcionalidad de acelerador hardware se realizaron diferentes pruebas. Inicialmente el sistema fue ejecutado de una manera muy básica en la cual se alinea una sola lectura de 20 nucleótidos a un genoma de referencia de 300 nucleótidos. La salida de dicha ejecución se muestra en el Listado 5.1. La muestra imprime en orden de aparición: el genoma de referencia al cual se alineará la lectura, el arreglo de sufijos (SA), la transformada de Burrows-Wheeler (TBW), la matriz de ocurrencia reducida (OccR) y el vector de frecuencias (C). Posteriormente pueden apreciarse los resultados de la estimación, en esta parte se observa como el software toma uno a uno los nucleótidos de la lectura (taacacacttcaaccgttac), calculando un nuevo intervalo parcial *k-l* en cada interacción, y reportando al procesar el último nucleótido, el

intervalo final de búsqueda que en este caso es 218-219, junto con la posición 40 que es en la cual aparece dentro del genoma de referencia. Posteriormente se imprimen tanto la TBW como la lectura codificada a dos bits, luego se muestra la concatenación de la TBW y la OccR y da comienzo la alineación hardware. Finalmente se presentan los resultados de la alineación hardware, como puede notarse el intervalo es idéntico al de la especulación software, lo cual muestra la correcta funcionalidad del sistema.

Listado 5.1 Salida del sistema al alinear una lectura.

```
dpachecob@dpachecob-G1-Sniper-B5:~/Tesis/DiseVivado/Aligner_v2/software$./Aligner_v1
~/Tesis/DiseVivado/Aligner_v1/firmware/Pico_Toplevel.bit
~/Tesis/DiseVivado/Aligner_v1/software/cromosoma
~/Tesis/DiseVivado/Aligner_v1/software/lecturas

================================================================
ACELERADOR HARDWARE DE LA ALINEACIÓN DE LECTURAS CORTAS DE ADN
Versión 2016.1
================================================================

Cargando el FPGA con
'/home/dpachecob/Tesis/DiseVivado/Aligner_v1/firmware/Pico_Toplevel.bit'


GENOMA DE REFERENCIA
Número de caracteres n=300
t a g g c g a t t t g a g t t a g g c a c c a c g g t c g c a t g c g c g c g a t a a
c a c a c t t c a a c c g t t a c g t g g g t t g g a g t t t a t t g a g c c c t c g
c t g c t g a a t a t c c g a t c g a t t g g c t t t t a g g c g t a c c g t g g c a
c t g g a c c c t g c a c c c g t t t t t t g c g c g a c c a a a c c g g t t t t g g t
t t t t g g g g g g g g a a a a c c g a t c g a t c g g g g t a t t t g c c c c t a t c
t t g a t a t g g g t a g g g c g c g g t t a a t c g t t g c c a c t c c g t t t t t
g c g c g a c c a a a c c g g t t t t t t g g g g g g g g a a a a c c g a t c g a t c

ARREGLO DE SUFIJOS
300 286 183 287 184 158 266 41 288 185 159 267 51 92 237 42 44 155 263 19 140 133 289
186 160 268 120 52 22 58 247 128 46 78 15 1 114 226 11 69 39 93 218 297 95 293 190
100 194 238 212 30 220 74 104 6 202 299 157 265 50 43 18 139 21 246 127 45 29 156 264
20 245 207 141 208 80 134 290 187 97 161 269 121 53 142 250 209 81 135 153 261 37 295
291 188 98 192 102 4 27 151 259 35 33 230 84 196 23 232 162 270 117 122 59 54 240 143
251 210 248 82 89 136 86 129 47 214 109 285 182 91 154 262 132 77 10 68 38 217 296
292 189 99 193 103 5 17 138 126 28 244 206 79 152 260 36 3 150 258 34 32 229 231 116
88 85 108 284 181 131 67 16 125 2 228 115 107 283 180 227 282 179 281 178 280 177 279
176 197 222 198 62 223 199 24 233 63 163 169 271 118 224 200 25 123 60 234 55 12 241
64 70 164 144 252 170 272 40 236 119 57 14 0 113 225 94 211 219 73 201 298 49 96 249
294 191 101 26 83 195 239 213 90 76 9 216 137 243 205 149 257 31 87 130 66 124 106
278 175 221 61 168 235 56 13 112 72 48 75 8 215 242 204 148 256 65 105 277 174 167
111 71 7 203 147 255 276 173 166 110 146 254 275 172 165 145 253 274 171 273

TRANSFORMADA DE BURROWS-WHEELER
c g g a a c c t a a a a c g t a c g g c c g a a a a t a c t c c c g t t t t g g g a g
g t g g g g a t c t t g g t t c c t a g g c c g a g a a a g g a c g a a a t a a a a c
t c c c g g g t c c c t t g t g g g g g t t a g c c g c a c t c c c a c g c g a a t g
g g t c c g t t g c t c c c c c c c c g t g c t t a c c c g t t c t g c g t c g g g t t
a t a g a t g g a g g g g g g t t c t g t g g c c g c t c c g g g c c g c a c g a g c
c g g a t g t t $ t g a c a t g a t a c a a a g c a a a c t t t c t t t t a c c t g t
t t a g t g g g t t c a t c g t t t g a t t t t g a a t t t t t c t t t t g g g t g g

MATRIZ DE OCURRENCIAS REDUCIDA
0       0       0       0
15      14      20      15
```

```
32      33      38      25
37      51      61      43
51      69      76      59

VECTOR DE FRECUENCIA C
1 57 129 218

ALINEACIÓN SOFTWARE
Procesando lectura: taacacacttcaaccgttac
Nucleótido actual: c
startblocK=0          charsblocK=0
startblocL=256        charsblocL=3   Intervalo parcial: 57-129
Nucleótido actual: a
startblocK=0          charsblocK=14
startblocL=128        charsblocL=0   Intervalo parcial: 15-33
Nucleótido actual: t
startblocK=0          charsblocK=2
startblocL=0          charsblocL=4   Intervalo parcial: 220-222
Nucleótido actual: t
startblocK=192        charsblocK=3
startblocL=192        charsblocL=4   Intervalo parcial: 264-265
Nucleótido actual: g
startblocK=256        charsblocK=3
startblocL=256        charsblocL=4   Intervalo parcial: 208-209
Nucleótido actual: c
startblocK=192        charsblocK=7
startblocL=192        charsblocL=8   Intervalo parcial: 115-116
Nucleótido actual: c
startblocK=64         charsblocK=13
startblocL=64         charsblocL=14  Intervalo parcial: 84-85
Nucleótido actual: a
startblocK=64         charsblocK=11
startblocL=64         charsblocL=12  Intervalo parcial: 27-28
Nucleótido actual: a
startblocK=0          charsblocK=11
startblocL=0          charsblocL=12  Intervalo parcial: 12-13
Nucleótido actual: c
startblocK=0          charsblocK=3
startblocL=0          charsblocL=4   Intervalo parcial: 60-61
Nucleótido actual: t
startblocK=0          charsblocK=14
startblocL=0          charsblocL=15  Intervalo parcial: 232-233
Nucleótido actual: t
startblocK=192        charsblocK=7
startblocL=192        charsblocL=8   Intervalo parcial: 268-269
Nucleótido actual: c
startblocK=256        charsblocK=0
startblocL=256        charsblocL=1   Intervalo parcial: 126-127
Nucleótido actual: a
startblocK=64         charsblocK=16
startblocL=64         charsblocL=17  Intervalo parcial: 32-33
Nucleótido actual: c
startblocK=0          charsblocK=10
startblocL=0          charsblocL=11  Intervalo parcial: 67-68
Nucleótido actual: a
startblocK=64         charsblocK=0
startblocL=64         charsblocL=1   Intervalo parcial: 16-17
Nucleótido actual: c
startblocK=0          charsblocK=4
startblocL=0          charsblocL=5   Intervalo parcial: 61-62
Nucleótido actual: a
startblocK=0          charsblocK=14
startblocL=0          charsblocL=15  Intervalo parcial: 15-16
Nucleótido actual: a
startblocK=0          charsblocK=6
startblocL=0          charsblocL=7   Intervalo parcial: 7-8
Nucleótido actual: t
startblocK=0          charsblocK=0
startblocL=0          charsblocL=1   Intervalo parcial: 218-219
```

```
La lectura actual se encuentra en el intervalo 218-219
Posición (es) en el genoma de referencia: 40


INICIANDO LA ALINEACIÓN HARDWARE

TBW COMPACTADA
3900d429  5d300969  aba2aff9  a35faf72  24a0225  5ea57403  58faabbd  c2645746
55dbe5ea  f953db95  8cfa9e6d  b7eaa8ac  96a5d96b  feca5891  60132c4b  d4ff7f40
71fab8fe  fc2ff2fe  2babfdf  0  0  0

LECTURA COMPACTADA
11f416f1 c1 0 0 0 0 0 0

CONCATENACIÓN
0       0       0       0       3900d429        5d300969        aba2aff9        a35faf72
f       e       14      f       24a0225         5ea57403        58faabbd        c2645746 20
        21      26      19      55dbe5ea        f953db95        8cfa9e6d        b7eaa8ac 25
        33      3d      2b      96a5d96b        feca5891        60132c4b        d4ff7f40 33
        45      4c      3b      71fab8fe        fc2ff2fe        2babfdf         0


Enviando la matriz de ocurrencia comprimida y la TBW a DDR3…
Enviando el comandos de configuración a registros del FPGA
Procesando lectura 1…
Esperando resultados...

La lectura actual se encuentra en el intervalo 218-219
Posición (es) en el genoma de referencia: 40


¡Todos los resultados coinciden con la estimación software!
```

Esta misma ejecución fue simulada en SWsim para verificar el comportamiento de las señales dentro del FPGA. Los resultados son las formas de onda en las figuras 5.1 y 5.2, en estas, los estados de la unidad de control se codifican como: INICIO=0, CONFIGURACIÓN=1, CARGA_LECTURA=2, DIRECCIONA_DDR1=3, DIRECCIONA_DDR2=4, ESPERA_DDR1=5, ESPERA_DDR2=6, CALCULA_SIGUIENTE=7,8 Y 9, ENVIA_RESULTADOS=10. En la Figura 5.1 se observa la recepción de los datos, puede notarse como la habilitación en alto de la señal *s1i_valid* habilita secuencialmente las señales *load_vectorC*, *load_confi*, y *load_read*, cargando en los registros correspondientes del diseño el vector de ocurrencias, el vector de configuración y la lectura a procesar, esto ocurre al final de los estados 0, 1 y 2 respectivamente. En los estados 3 y 4 puede observarse como la señal c0_s1_axi_arvalid se pone en alto para enviar las direcciones a memoria. Note también el cambio de la señal *sw_k*/ para colocar las direcciones apropiadas en la señal co_s1_axi_araddr en esos mismos estados. Posteriormente, en el estado 5 comienza la espera de los datos de memoria. Observe que hasta ahora, la señal *ini* se encuentra deshabilitada, por lo que *k* y */* contienen los valores iniciales del algoritmo (0 y 301 respectivamente).

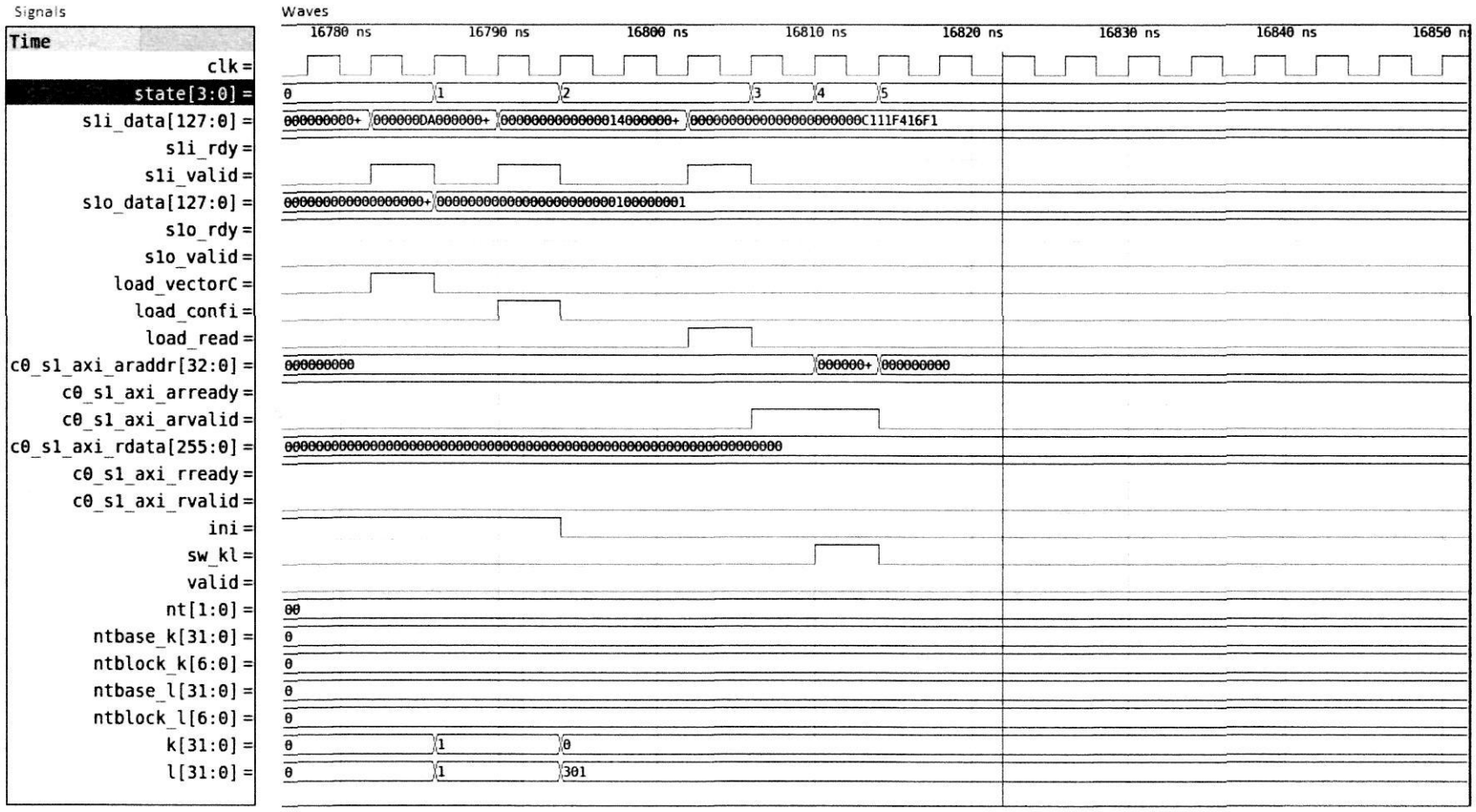


Figura 5.1 Formas de onda de la simulación funcional del diseño. Primeros estados de transición.

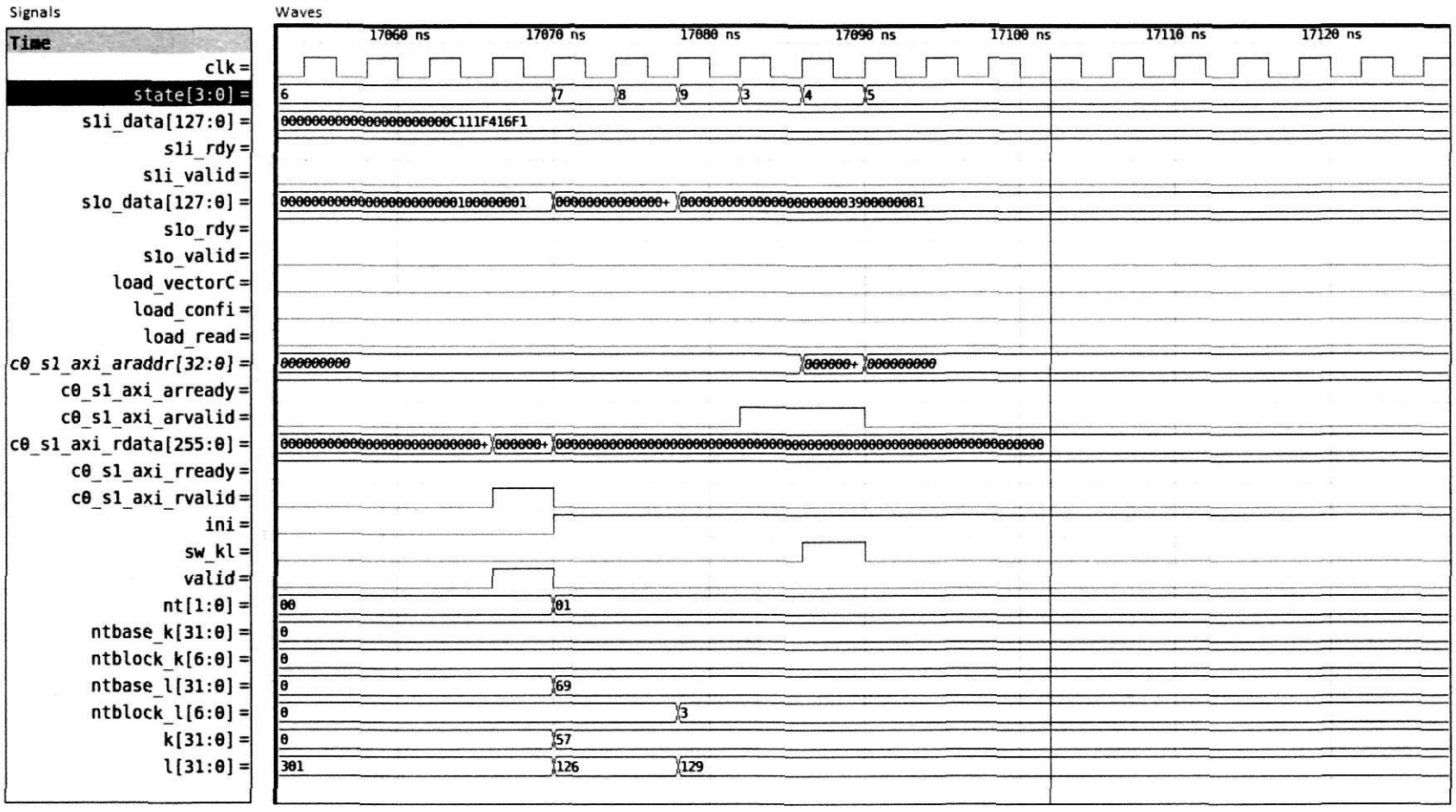


Figura 5.2 Formas de onda de la simulación funcional del diseño. Siguientes estados de transición

La figura 5.2 muestra como la señal c0_s1_axi_arvalid se pone en alto al final del estado 6, lo que indica que se encuentra disponible el segundo dato proveniente de memoria. Al realizarse la transición al estado 7, el registro de lecturas se desplaza a la derecha y la señal *nt* (la cual es su salida), contiene ahora el código 01, que corresponde a la Citosina (el último elemento de la lectura y primero a procesar), todos los demás valores también se actualizan y después de tres ciclos, al final del estado 9, *k* y *l* han adquirido sus nuevos valores, 57 y 129, lo cual coincide con la estimación software en su primera iteración. Note como *k* y *l* tomaron el valor del cálculo puesto que la señal *ini* ahora está en alto. Posteriormente la máquina vuelve al estado 3 para procesar un nuevo nucleótido.

La versión anterior es solamente demostrativa por lo que se modificó para evitar los tiempos de impresión en pantalla de los índices de FM, de esta forma se tiene una estimación más cercana a la realidad de los tiempos de procesamiento software y hardware que se presentan en la siguiente sección. A continuación se muestra la salida del sistema al alinear 35 lecturas de 45 nucleótidos al cromosoma 21. Note como en esta ejecución algunas lecturas no existen en la referencia y el algoritmo abandona la búsqueda en forma prematura cuando el valor de *k* es igual al valor de *l*. Por otra parte, algunas ocurren en varias posiciones y el sistema es capaz de identificar todas estas. Finalmente, observe como los resultados de la alineación hardware coinciden al 100% con la estimación software.

Listado 5.2. La salida del sistema al alinear múltiples lecturas al cromosoma 21.

```
dpachecob@dpachecob-G1-Sniper-B5:~/Tesis/DiseVivado/Aligner_v2/software$ ./Aligner_v1
~/Tesis/DiseVivado/Aligner_v2/firmware/Pico_Toplevel.bit
~/Tesis/DiseVivado/Aligner_v2/software/chr21.fa
~/Tesis/DiseVivado/Aligner_v2/software/r0_chr21.fq

================================================================
ACELERADOR HARDWARE DE LA ALINEACIÓN DE LECTURAS CORTAS DE ADN
Versión 2016.1
================================================================

Cargando el FPGA con el archivo
'/home/dpachecob/Tesis/DiseVivado/Aligner_v2/firmware/Pico_Toplevel.bit' ...
Leyendo el genoma de referencia
'/home/dpachecob/Tesis/DiseVivado/Aligner_v2/software/chr21.fa' ...
Número de caracteres leidos: n=48129895
Obteniendo los índices de FM...

INICIANDO LA ALINEACIÓN SOFTWARE
No      Lectura                                     Intervalo                Posición(es)
1       tcacttagaggattcatgctgggattgaatagtaggaccacacat    0027378737-0027378738    0017526450
2       gaaattttaagcagaaatgctttagaagaaatagtctttatttat    0017848139-0017848140    0017526500
3       tttttaaaagagcttttaaattactgagttctaatacagaagaag    0025311917-0025311917    No encontrada
4       agtttgaattcatggcgtattgctataatgcactattagactaat    0007645959-0007645960    0017526600
5       attttcttgtcaattaaagacttacctttattcactgattcactt    0010338415-0010338416    0017526650
6       ccccttccttttttgaataattctccctgcccagtgactcacccag   0013909197-0013909198    0042953150
```

7	gatgagacacagagctcatggttgcctcctgtgaagtgcaggaag	0019414825-0019414826	0042953200
8	agcagacagacccaaaagcactggcctcccagatgaatcaggagg	0006120133-0006120134	0042953250
9	gcaggctgggtgacgtctgccccggaaccaggaaagtgcggctgc	0019995120-0019995121	0042953300
10	ccacaccccgcccgccgcccaggacccagggcccttctgaaaagt	0013151192-0013151193	0042953350
11	gctctttggagtaactggtgctttctatcagttttcaagagcata	0020870186-0020870187	0016351532
12	acacaagcacaaggtcaaggtcacagatcagcaagaaagattacc	0003684600-0003684601	0016351750
13	cctggccaacatggtgaaatcctgtctctactaaaaatacaaaaa 0043602431 0025116749 0016352600	0014555372-0014555376	0026062557
14	tggacacagtggcatgagcctgtaatcccagctactcaggaggct	0030384027-0030384028	0016352650
15	acgagaatcgcttaaacctggaaggcggaggttgcagtgagtcaa	0002326218-0002326218	No encontrada
16	cactattgcactccagctgggtgacagagcgagactccgtatcaa	0011493461-0011493462	0016352750
17	aataaaataaaataaaaataaaataaatacatacatatatgcaat	0002589548-0002589549	0016352800
18	aacaaaatcatgccttttgttttagctgtttaaacaagatgttaa	0001379037-0001379038	0016352850
19	ttttaagttacatttatgaaaccaagagttatgtgatcaatgttc	0034599486-0034599487	0016352900
20	gaagcattctggtcaagaaatcattttgaaggatgcataaagctg	0018018019-0018018020	0016352950
21	gtaaatttacactataaaagctttccttgctaaatcaagtaccac	0023027005-0023027006	0016353000
22	gtagctttatacaatttgcaaactacttatgctatatgtacatgt	0023218075-0023218076	0016353050
23	ttctcaattagtggtttttcatatttttgataaattaaatacctt	0032890254-0032890255	0016353100
24	gaaaataacccaaaatttactatcctagaaagattgctagttttt	0017655527-0017655528	0016353150
25	ataaagtaaaaaagaaaaattatgtcaaattatcaaattaaatac	0007747090-0007747091	0016353200
26	aaatgacttatatgacatcactctagtcattacaggctccaaaca	0001191888-0001191889	0016353250
27	atatctagatagaagtttccagtcaagctgtggaagagaaatatc	0008272927-0008272928	0016353300
28	agacacaattcatgagcacccagacatggtggtgatacctgtgaa	0005553884-0005553885	0016353350
29	aataaatctgactaggtcccatcctacaggattaaggaatcaaaa	0002615364-0002615365	0016353400
30	aatactacatgcacaatgtacctgcagaaattctatgtctttaaa	0002708110-0002708111	0016353450
31	acaggtaacattaaagtatgtctgtcaacttggaaaatgaagcta	0003958700-0003958701	0016353500
32	acaccatcagcaggccacttacttacttaaatggttttatttat	0003756668-0003756669	0016353550
33	ttgaagaaactgcggtgcgtgctttaccctgatctcttatttgcc	0033139650-0033139651	0016353600
34	atccatacaaccaagtgtttgaagtactaatctcaggaacaatag	0008610069-0008610070	0016353650
35	*aaatgctaatctgagaaccagttctgctgttgtccaggtttaggg*	*0001215043-0001215044*	0016353700

Lecturas procesadas: 35

INICIANDO LA ALINEACIÓN HARDWARE

Comprimiendo las lecturas y la TBW...
Enviando la matriz de ocurrencia comprimida y la TBW a DDR3
Enviando el comandos de configuración a registros del FPGA

No	Lectura	Intervalo	Posición(es)
1	cb285113d39ea3e0347c8a30	0027378737-0027378738	0017526450
2	cb7f3f380e7f208203fc240	0017848139-0017848140	0017526500
3	dc312082ff03c78b3ff00890	0025311917-0025311917	No encontrada
4	91cf21c3a6cf9cc3bf83d30	0007645959-0007645960	0017526600
5	3d1e3d1ff021f17fff7ed00	0010338415-0010338416	0017526650
6	52e1d152830f7579157d7ff0	0013909197-0013909198	0042953150
7	ee0b92829d3af97523884480	0019414825-0019414826	0042953200
8	48e0d28a91e9759212150	0006120133-0006120134	0042953250
9	280b9a796de55a0524a7ab80	0019995120-0019995121	0042953300
10	a57de00b9654a15214455950	0013151192-0013151193	0042953350
11	2ff4224c1eb9fdcd277fa2c0	0020870186-0020870187	0016351532
12	242023c5b42b448d4424420	0003684600-0003684601	0016351750
13	1c00c400b80d7b7717a504e0 0043602431 0025116749 0016352600	0014555372-0014555376	0026062557
14	271d28a73897b0d53a112e90	0030384027-0030384028	0016352650
15	af92e2d017a0a68620d9f	0002326218-0002326218	No encontrada
16	621d6cd049eae12211cf91d0	0011493461-0011493462	0016352750
17	4c4cce43300c0c300c030	0002589548-0002589549	0016352800
18	f0108ef07feff27b100d390	0001379037-0001379038	0016352850
19	cee343bdf380508b3fc2f130	0034599486-0034599487	0016352900
20	28e4c09e42034ff82093deb0	0018018019-0018018020	0016352950
21	70342c513009fd7e2c0fc470	0023027005-0023027006	0016353000
22	e733b13b3f901c7c2c9fcc40	0023218075-0023218076	0016353050
23	303c0c5fbff4cffe3dd0f2e0	0032890254-0032890255	0016353100
24	8f9cbfffc735c8200c1500	0017655527-0017655528	0016353150
25	f340f031800f3b40c0b0000	0007747090-0007747091	0016353200
26	f129d40484d1dcb4e1f330	0001191888-0001191889	0016353250
27	ba0880cd2fd4b427cdc8c80	0008272927-0008272928	0016353300
28	eb8c5ee08915213a8443d30	0005553884-0005553885	0016353350
29	8f0a0d002b54d7123037870	0002615364-0002615365	0016353400
30	3dcedfc043b1792031c4e40	0002708110-0002708111	0016353450
31	fa00e09cb3b7b414ac13c0	0003958700-0003958701	0016353500
32	3aff3f3947c7c7f4534920	0003756668-0003756669	0016353550
33	e377cfe5ae6e7f153e080790	0033139650-0033139651	0016353600
34	374a04322efe0b1cd4c4140	0008610069-0008610070	0016353650
35	fb52bf2a2052f79ee70de0	0001215043-0001215044	0016353700

Lecturas procesadas: 35
Porcentaje de coincidencias Hardware-Software: 100%
¡Todas las pruebas exitosas!

5.2 Razón de procesamiento

Posteriormente el sistema fue probado alineando diferentes conjuntos de lecturas cortas a diversos cromosomas del genoma humano. Las lecturas fueron extraídas artificialmente de dichas cadenas mediante el programa Gsim. Este programa genera lecturas en formato FASTQ, permitiendo simular diversos parámetros como los errores de secuenciación, la razón de mutaciones, y la longitud y número de lecturas. Además, en el archivo de salida genera información del origen de la lectura, lo que permite comprobar si las alineaciones son correctas.

En la tabla 5.1 se presentan los resultados al alinear conjuntos de 1 000 000 de lecturas de 30, 45 y 60 nucleótidos al cromosoma 21. En cada caso se realizaron 20 corridas, variando los parámetros en Gsim y promediando los resultados para obtener un valor significativo. El tiempo de pre-procesamiento se refiere al tiempo para calcular los índices de FM y algunas otras operaciones de manejo de archivos. El factor de aceleración se calcula a partir de la ley de Amdahl. La concordancia software-hardware se refiere al porcentaje de lecturas cuyos resultados de alineación hardware coinciden con los de la alineación software. Las lecturas alineadas a su posición original, fueron determinadas al comparar los resultados de alineación hardware con el origen de la lectura proporcionado por la herramienta Gsim. El significado de los otros parámetros no requiere explicaciones adicionales.

Los datos demuestran una eficacia del 100%, al obtener igual porcentaje de concordancia hardware-software y al comprobar los resultados con la información de origen proporcionada por Wgsim. También se observa un factor de aceleración hardware promedio de 1.34 en relación a la ejecución software, pero que muestra un ligero incremento al aumentar la longitud de la lectura. La comparación de velocidad es válida puesto que ambas implementaciones se ejecutan en el mismo sistema anfitrión.

Tabla 5.1 Resultados de la alineación de 1 000 000 de lecturas al cromosoma 21.

PARAMETRO	PRUEBAS		
Longitud de lectura	40 nucleótidos	50 nucleótidos	60 nucleótidos
Tiempo de pre-procesamiento	7.917625189 s.	7.931704044 s.	7.856386185 s.
Tiempo de alineación software	213.0899858 s.	237.0632579 s.	260.0875359 s.
Tiempo de alineación hardware	167.1810338 s.	176.1040361 s.	187.1264360 s.
Factor de aceleración	1.27	1.35	1.39
Concordancia Software-Hardware	100%	100%	100%
Lecturas alineadas a su posición original	100%	100%	100%

La prueba se modificó para caracterizar el comportamiento del acelerador al incrementar la longitud de la cadena de referencia. Esta vez se alinearon conjuntos de 1 000 000 lecturas con longitud de 60 nucleótidos a los cromosomas: 22, 16, 11, 6, 3 y 1, obteniendo los resultados de la Tabla 5.2. Como es de esperarse el tiempo de pre-procesamiento se incrementa al generar índices de mayor tamaño. Sin embargo al observar los tiempos de procesamiento software y hardware, se nota un crecimiento exponencial en el primero, mientras que el segundo se mantiene prácticamente constante, como resultado el factor de aceleración se incrementa. Esto significa que a medida que la longitud de la referencia crece, el acelerador hardware se desempeña con mayor eficiencia, logrando un factor de 6.86 cuando se alinea al cromosoma humano más grande.. Lo anterior puede observarse con mayor claridad en las gráficas de las Figura 5.3 y 5.4.

Tabla 5.2 Resultados de la alineación de 1 000 000 de lecturas a diversos cromosomas.

PARAMETRO	PRUEBAS					
Cadena de referencia	Chr22	Chr16	Chr11	Chr6	Chr3	Chr1
Long. de la referencia	49 528 953	88 822 254	132 130 853	170 896 993	199 446 827	247 199 719
T. de pre-proc.	8.14119	17.70956	28.94125	40.07135	175.15554	246.28346
T. de alineación S.	260.69301	262.88986	282.06603	539.18839	614.64447	1372.88544
T. de alineación H.	186.47966	187.00379	196.70732	200.83575	205.05696	198.05066
Factor de aceleración	1.39	1.40	1.44	2.68	2.99	6.82
Concordancia H-S	100%	100%	100%	100%	100%	100%
Lecturas alineadas a su posición original	100%	100%	100%	100%	100%	100%

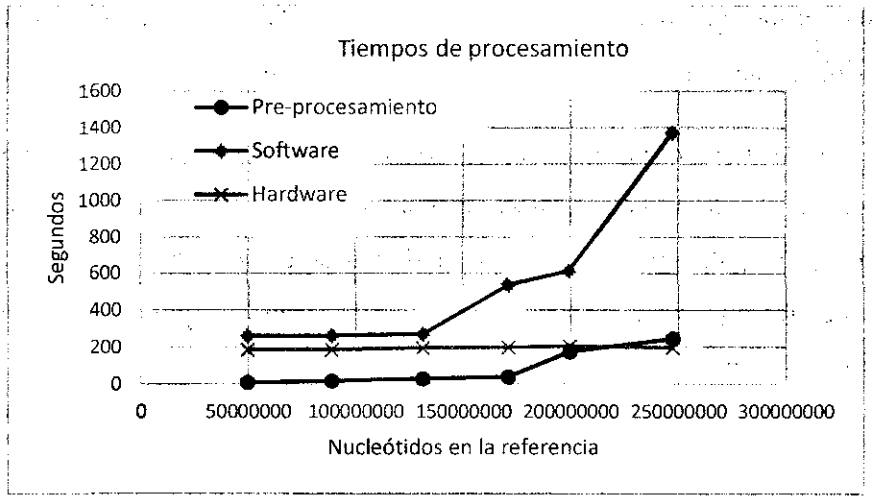


Figura 5.3 Tiempos de procesamiento del sistema.

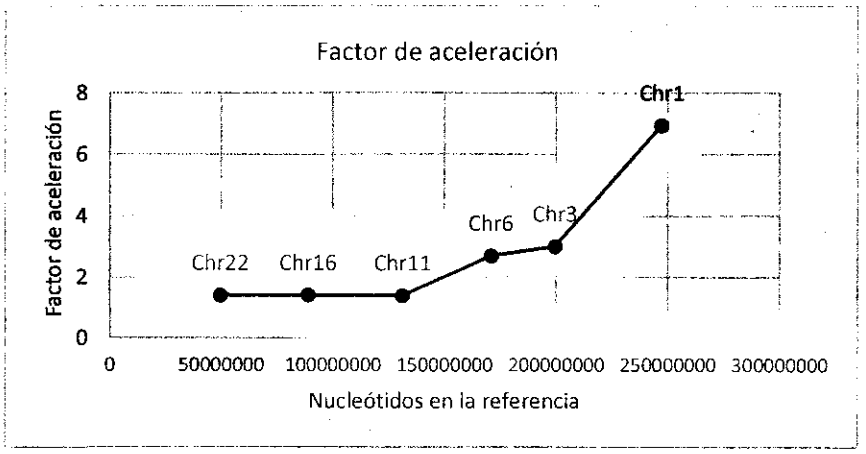


Figura 5.4 Factor de aceleración del sistema.

5.3 Uso de área en el chip y consumo de potencia.

La cantidad de área utilizada en el FPGA es otro factor fundamental del diseño, puesto que de esto depende el número de núcleos de alineación que pueden albergarse en un chip para maximizar el paralelismo de la aplicación y por consiguiente la razón de procesamiento. Los recursos utilizados en el diseño se muestran en la Figura 5.5, como puede observarse solo los bloques de entrada y salida (I/0), los transceptores (GT), los buffers (BUFG) y los módulos de manejo del reloj (MMCM) superan el 20% de su totalidad. Estos bloques se utilizan principalmente en el Pico Framework, por lo que no

incrementan al incorporar más núcleos de alineación. Bajo este principio se determina que es posible incluir hasta 5 núcleos de alineación en el diseño.

Por su parte la Figura 5.6 muestra el consumo de potencia en el FPGA, y su distribución en los diferentes módulos y señales. Siendo el consumo total de 5.45W proveniente principalmente de la potencia dinámica del circuito. Ambas estadísticas (consumo de área y potencia), fueron extraídas de los reportes post-implementación de la herramienta Vivado.

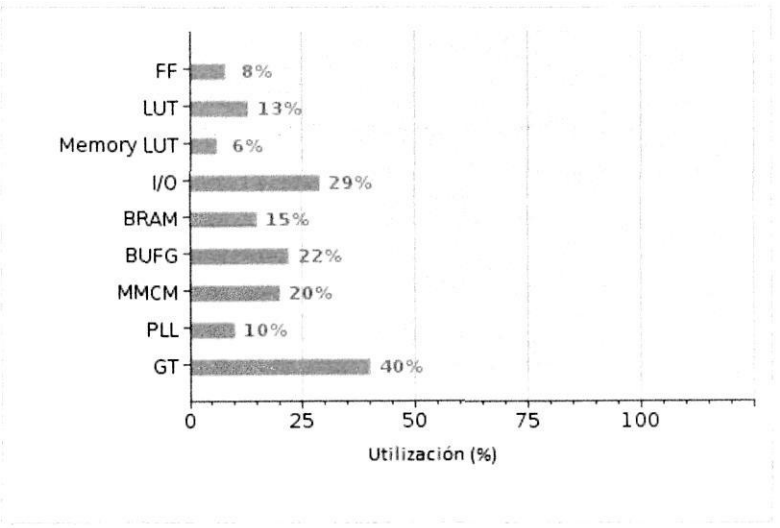


Figura 5.5 Utilización de recursos del FPGA.

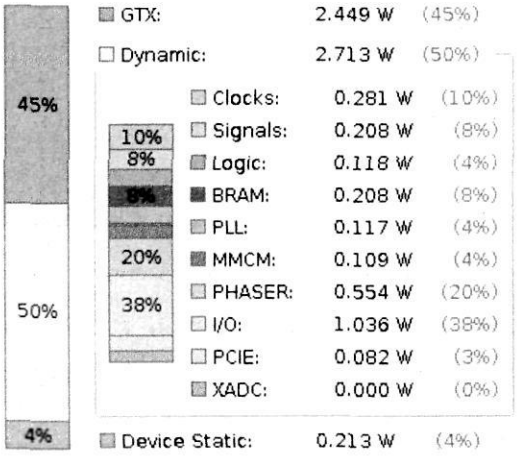


Figura 5.6 Distribución del consumo de potencia en el Firmware.

5.4 Estimación para múltiples núcleos de alineación

Las pruebas presentadas únicamente han considerado un núcleo de alineación, sin embargo, debido a la modularidad del mismo, este puede expandirse para albergar múltiples núcleos. En la sección anterior se estimó que el FPGA incluido en la tarjeta podría albergar hasta 5 módulos de alineación, por lo que teóricamente la velocidad se incrementaría en el mismo factor, lamentablemente esto no ocurre debido a que únicamente se cuenta con un controlador de memoria, aun así las lecturas pueden segmentarse, en el contexto segmentar la memoria significa enviar direcciones de lectura al controlador sin esperar a que el resultado de la primera lectura retorne, esto le permite ordenar apropiadamente las peticiones y disminuir los tiempos de acceso promedio. De esta forma al incluir los 5 módulos de alineación segmentando apropiadamente los accesos puede lograrse, de acuerdo a las especificaciones de la tarjeta, una mejora en la velocidad en un factor aproximado de 3. Debe enfatizarse que este calculó se realiza para la tarjeta M505k325, pero al utilizar alguna de mayor número de controladores de memoria, el factor puede ser mucho mayor. Adicionalmente pueden instalarse hasta 6 módulos M505 en el *backplane* M500 trabajando simultáneamente, el factor de aceleración con esta mejora en combinación con la anterior seria cercana a 15 (considerando la sobrecarga de trabajo) con respecto al diseño base de un solo núcleo.

Generalmente los nuevos programas de alineación toman como referente a BWA, por lo que se realizaron las mismas pruebas de alineación de la sección 5.2 en dicho programa y se comparó con la estimación anterior. Los resultados se muestran en la Tabla 5.3, note como la implementación a su máxima capacidad puede superar en velocidad a BWA en un factor de aproximadamente 50X. Mediante este referente también se comparó en la Tabla 5.4 el desempeño del acelerador propuesto con los trabajos relacionados de la sección 2.6. Puede notarse como el diseño, utilizando un solo núcleo, es comparable en velocidad con CUSHAW Y SOAP 3, y a su máxima capacidad compite con SHEPARD y el trabajo de Fernández, sin embargo utiliza mucha menor memoria que el primero y no tiene la limitante del segundo en relación al procesamiento de solo unos cuantos KiB de nucleótidos.

Tabla 5.3 Estimación del desempeño del sistema con múltiples núcleos.

Sistema	Tiempo de Pre-procesamiento (s)	Tiempo de alineación (s)
BWA	192.56321	650.17225
Acelerador base	246.28346	198.05066
Acelerador con 5 núcleos.	246.28346	66.016886
Acelerador con 6 módulos	246.28346	13.203377

Tabla 5.4 Comparación de resultados con trabajos relacionados.

Alineador	Tipo de acelerador	Algoritmo base	Características
SOAP3 (Liu C. M., y otros, 2012)	GPU	Tablas Hash	Sus creadores reportan aceleraciones de 7.5X comparado a BWA. Su diseño no soporta alineación con espacios.
CUSHAW (Liu, Schmidt, & Maskell, 2012)	GPU	Índices de FM	6X comparado a la versión multi-hilos de BWA. No permite alineaciones con espacios
SHEPARD (Nelson, Towsend, Rao, Jones, & Zambreno, 2012)	FPGA Sistema Convey HC1	Tablas Hash	Reportan velocidades extremadamente altas comparadas con herramientas software como BWA, Bowtie y MAQ (arriba de 100x). Soporta únicamente alineación exacta y es capaz de alinear solo un bajo porcentaje de las lecturas entrantes, además de utilizar una elevada cantidad de memoria (22 GB)
Trabajo de Fernández. (Fernández, Najjar, & Lonardi, 2011)	FPGA	Índices de FM	Al comparar sus resultados con el software de alineación BOWTIE, obtienen aceleraciones mayores a 100X, sin embargo al almacenar la tabla de ocurrencia y el vector de frecuencia en módulos BRAM, su aplicación se limita a cadenas de ADN relativamente cortas.
Este trabajo	FPGA	Índices de FM	Se estima una aceleración de 50X con respecto a BWA. No permite alineaciones inexactas. Requiere menos de 3 GB de memoria en la tarjeta de desarrollo.

6 CONCLUSIONES Y TRABAJO FUTURO

En esta tesis se investigó el estado del arte de las tecnologías de secuenciación de ADN y de los programas de alineación de lecturas cortas, encontrando que dichos programas siguen siendo muy lentos comparados con la enorme velocidad de los secuenciadores actuales, convirtiéndose en el cuello de botella del proceso de análisis de genomas. Posteriormente se determinó que la mayoría de estos programas están basados en una de dos estrategias: Tablas Hash o Transformada de Burrows-Wheeler (TBW), ambas compartiendo el principio de pre-procesar la información creando una estructura (una tabla en el primer caso y un índice en el segundo) que facilite, en el paso complementario, la localización de la lectura en la referencia. No obstante, desde el punto de vista hardware la segunda solución resulto más atractiva, por varias razones entre las que destacan el menor uso de memoria, la relativa facilidad de incorporar búsquedas inexactas al algoritmo sin modificar la estructura de indexado inicial, así como las operaciones simples requeridas, las cuales lo hicieron ideal para su implementación en FPGAs.

De esta forma se diseñó una arquitectura para implementar el algoritmo de búsqueda exacta basadas en los índices de FM, la cual es completamente modular y escalable. Su implementación hace uso de diferentes técnicas tales como el almacenar solo ciertas filas del arreglo de ocurrencias y recuperar el resto en tiempo real a partir de la TBW de la referencia, y del intercalado de memoria, reflejando una importante reducción de los requisitos de memoria del sistema y del número de accesos a esta.

El diseño fue sintetizado e implementado en el entorno de desarrollo vivado 2014.2 y probado en la tarjeta de aceleración M505k325t de la empresa Pico Computing, la cual contiene el FPGA Xilinx Kintex-7 k325t. La tarjeta fue instalada en una computadora con procesador Intel Core I5 de cuarta generación con 8 GB en memoria DDR. Los resultados comprobaron que el algoritmo se acelera utilizando este tipo de arquitectura alterna, encontrando un factor de aceleración de 6.82X con respecto a su ejecución software al incorporar un solo núcleo de alineación y procesar el cromosoma 1 (247 199 719 nucleótidos), e incrementa exponencialmente al aumentar el número de nucleótidos en la cadena de referencia. Adicionalmente, las características de modularidad del diseño permiten estimar un factor cercano a 50X con respecto a BWA en una implementación de múltiples núcleos de alineación que explote completamente la capacidad del sistema, la cual es de hasta 6 tarjetas M505 trabajando con el CPU anfitrión.

La confirmación de la estimación previa queda pendiente para trabajo futuro junto con la inclusión de la alineación inexacta, la cual se sugiere sea implementada mediante la técnica siembra y extiende aplicada en el algoritmo de búsqueda basados en tablas Hash. En esencia se utilizará la implementación de alineación exacta descrita en este trabajo para encontrar las lecturas que alinean en forma exacta a la referencia y su posición en esta. Posteriormente las lecturas no encontradas serán divididas en semillas y con el mismo algoritmo se localizarán las posibles áreas de mapeo. Finalmente, se extenderá la búsqueda alrededor del área probable mediante un algoritmo de alineación local, tal como el Smith-Waterman, el cual reemplazará a los alineadores originales haciendo uso de reconfiguración dinámica parcial en el FPGA.

Otra sugerencia para trabajo a futuro es el uso de tarjetas de aceleración que incluyan múltiples controladores de memoria. Al hacerlo la velocidad del sistema puede incrementarse notablemente incluso si solo se utiliza un FPGA. En tal caso el núcleo de alineación propuesto únicamente requerirá un tiempo de acceso para obtener los datos necesarios para el cálculo de los valores de k y l en cada interacción, multiplicando inmediatamente la velocidad del núcleo por 2, sin requerir segmentar los accesos a memoria.

REFERENCIAS

Altschul, S., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology, 215*(3), 403-410.

Arram, J., Tsoi, K. H., Luk, W., & Jiang, P. (2013). Reconfigurable acceleration of short mapping. *21st Annual International IEEE Symposium on Field-Programmable Custom Computing Machines* (págs. 210-217). Seattle, WA: IEEE.

Avery, O., MacLeod, C., & McCarty, M. (1944). Studies on the chemical nature of the substance inducing transformation of pneumococcal types. *Journal of experimental medicine, 79*, 137-158.

Bentley, D. R., Balasubramanian, S., Swerdlow, H. P., Smith, G. P., Milton, J., Bromn, C. G., y otros. (2008). Accurate whole human genome sequencing using reversible terminator chemistry. *Nature, 456*, 53-59.

Burrows, M., & Wheeler, D. J. (1994). *A block sorting lossless data compression algorithm.* Reporte Técnico, Digital Equipment Corporation, Systems Research Center, Palo Alto, California.

Campagna, D., Albiero, A., Bilardi, A., Caniato, E., Forcato, C., Manavski, S., y otros. (2009). PASS: a program to align short sequences. *Bioinformatics, 25*(7), 967-968.

Che, S., Li, J., Sheaffer, J. W., Skadron, K., & Lach, J. (2008). Accelerating compute-intensive applications with GPUs and FPGAs. *Application specific processors, SASP2008* (págs. 101-107). Anahem, CA: IEEE.

Cock, P. J., Fields, C. J., Goto, N., Heuer, M. L., & Rice, P. M. (2010). The sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Research, 38*(6), 1767-1771.

Dahm, R. (2005). Friedrich Miescher and the dicovery of DNA. *Developmental Biology, 278*(2), 274-288.

David, M., Dzamba, M., Lister, D., LLe, L., & Brudno, M. (2011). SHRiMP2. *Bioinformatics, 27*(7), 1011-1012.

Drmanac, R., Sparks, A. B., Callow, M. J., Halpern, A. L., Burns, N. L., Kermani, B. G., y otros. (2010). Human genome sequencing using unchained base reads on self-assembling DNA nanoarrays. *Science, 327*(5961), 78-81.

Eid, J., Fehr, A., Gray, J., Luong, K., Lyle, J., Otto, G., y otros. (2009). Real-time sequencing fron single polymerase molecules. *Science, 323*, 133-138.

Fakruddin, Mohammad Mazumdar, R., Chowdhury, A., Hossain, N., Mahajan, S., & Islam, S. (2013). Pyrosequencing-A Next generation sequencing technology. *World Applied Sciences Journal, 12*(24), 1578-1571.

Fernández, E., Najjar, W., & Lonardi, S. (2011). String matching in hardware using the FM-index. *IEEE International Symposium on field-programmable custom computing machines* (págs. 218-225). Salt Lake City, UT: IEEE.

Ferragina, P., & Manzini, G. (2000). Opportunistic data structures with applications. *Foundations of computer science* (págs. 390-398). Redondo Beach, CA: IEEE.

Fonseca, N. A., Rung, J., Brazma, A., & Marioni, J. (2012). Tools for mapping high-throughput sequencing data. *Bioinformatics, 28*(24), 3169-3177.

Frese, K. S., Katus, H. A., & Meder, B. (2013). Next-Generation Sequencing: From understanding biology to personalized medicine. *Biology, 2*, 378-398.

Guyton, A. C., & Hall, J. E. (2006). *Tratado de fisiología médica* (11 ed.). Elsevier.

Hauck, S., & DeHon, A. (2008). *Reconfigurable computing, the theory and practice of FPGA-based computation.* USA: Elsevier.

Homer, N., Merriman, B., & Nelson, S. (2009). BFAST: an alignment tool for large scale genome resequencing. *PLoS ONE, 4*.

Hui, P. (2012). Next generation sequencing: chemistry, technology and applications. *Top Curr Chem.*

Jiang, H., & Wong, W. (2008). SeqMap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics, 24*(20), 2395.

Kim, M., Clauson, C., Olson, C. B., Ebeling, C., Hauck, S., & Ruzzo, W. L. (2012). Hardware acceleration of short read mapping. *Annual symposium on field programmable custom computing machines* (págs. 161-168). Toronto. Ontario Canada: IEEE.

Knuth, D. E., Morris, J. H., & Pratt, V. R. (1977). Fast pattern Matching in Strings. *SIAM Journal on Computing, 6*(2), 323-350.

Langmead, B., Trapnell, C., Pop, M., & Salzberg, S. (2009). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology, 10*(3).

Lee, W. P., Stromberg, M. P., Ward, A., Stewart, C., Garrison, E. P., & Marth, G. T. (2014). Mosaik: A hash-based algorithm for accurate next-generation sequencing short-read mapping. *PLoS ONE, 9*(3).

Li, H., & Durbin, R. (2009). Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics, 25*(14), 1754-1760.

Li, H., & Homer, N. (2010). A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Biointormatics, 2*(5), 473-483.

Li, H., Ruan, J., & Durbin, R. (2008). Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Research, 18*, 1851-1858.

Li, I. T., Shum, W., & Truong, K. (2007). 160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA). *BMC Bioinformatics, 8*(185).

Li, R., Li, Y., Kristiansen, K., & Wang, J. (2008). SOAP: short oligonucleotide alignment program. *Bioinformatics, 24*(5), 713-714.

Li, R., Yu, C., Li, Y., Lan, T. W., Yiu, S. M., Kristiansen, K., y otros. (2009). SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics, 25*(15), 1966-1967.

Li, Z., Chen, Y., Mu, D., Yuan, J., Shi, Y., Zhang, H., y otros. (2012). Comparison of the two major classes of assembly algorithms. *Briefings in Functional Genomics, 11*(1), 25-37.

Lin, H., Zhang, Z., Zhang, M. Q., Ma, B., & Li, M. (2008). Zoom! Zillions of oligos mapped. *Bioinformatics, 24*(21), 2431-2437.

Liu, C. M., Wong, T., Wu, E., Luo, R., Yiu, S. M., Li, Y., y otros. (2012). SOAP3: Ultra-fast GPU-based parallel alignment tool for shor reads. *Bioinformatics advance.*

Liu, C.-M., Wong, T., Wu, E., Luo, R., Yiu, S.-M., Li, Y., y otros. (2012). SOAP3: Ultra-fast GPU-based parallel alignment tool for short reads. *Bioinformatics advance access published, 28*(6), 878-879.

Liu, L., Li, Y., Li, S., Hu, N., He, Y., Pong, R., y otros. (2012). Comparison of Next generation sequencing systems. *Journal of Biomedicine and biotechnology*, 1-11.

Liu, Y., Schmidt, B., & Maskell, D. L. (2012). Cushaw:a cuda compatible short read aligner to large genomes based on the burrows-wheeler transform. *BMC research notes, 5*(1), 27.

Margulies, M., Egholm, M., Altman, W. E., Attiya, S., Bader, J. S., Bemben, L. A., y otros. (2005). Genome sequencing in microfabricated high density picolitre reactors. *Nature, 437*, 376-380.

Maxam, A., & Gilbert, A. (1977). A new method for sequencing DNA. *PNAS, 74*(2), 560-564.

Miller, J. R., Koren, S., & Sutton, G. (2010). Assembly algorithms for next-generation sequencing data. *Genomics, 95*(6), 315-327.

Muse, S. (2005). Genomics and bioinformatics. En J. D. Enderle, S. M. Blanchard, & J. D. Bronzino, *Introduction to biomedical engineering* (2 ed., págs. 799-831). Elsevier.

Myllykangas, S., Buenrostro, J., & Ji, H. P. (2012). Overview of sequencing technology platforms. En N. Rodríguez Ezpeleta, M. Hackenberg, & A. M. Aransay, *Bioinformatics for high throughput sequencing* (págs. 11-25). Springer.

Navarro, G., Baeza-Yates, R., Sutinen, E., & Tarhio, J. (2001). Indexing methods for aproximate string matching. *IEEE Data Engineering Bulleting, 24*(4), 19-27.

Needleman, S. B., & Wunsch, C. D. (1970). A generl method applicable to the search for similarities in the aminoacid sequence of two proteins. *Journal of Molecular Biology, 48*(3), 443-453.

Nelson, C., Towsend, K., Rao, B. S., Jones, P., & Zambreno, J. (2012). Shepard: A fast exact match short read aligner. *Formal Methods and Models for Codesign (MEMOCODE),2012 10th IEEE/ACM International Conference on*, (págs. 91-94).

Ning, Z., Cox, A., & Mullikin, J. (2001). SSAHA: A fast search method for large DNA databases. *Genome Research, 11*(10), 1725-1729.

Pelak, K., Shianna, K. V., Ge, D., Maia, J. M., Zhu, M., Smith, J. P., y otros. (2010). The characterization of twenty sequenced human genomes. *PLoS Genetics, 6*(9).

Perkel, J. M. (Febrero de 2014). *Next-Gen Sequencing 2014 Update.* Recuperado el 5 de Septiembre de 2014, de http://www.biocompare.com/Editorial-Articles/155411-Next-Gen-Sequencing-2014-Update/

Pop, M. (2004). Shotgun sequence assembly. *Advances in computers, 60*, 193-248.

Quail, M. A., Smith, M., Coupland, P., Otto, T. D., Harris, S. R., Connor, T. R., y otros. (2012). A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers. *BMC Genomics, 13*(341).

Rizk, G., & Lavenier, D. (2010). GASSST: Global alignment short sequence search tool. *Bioinformatics, 26*(20), 2534-2540.

Ruffalo, M., LaFramboise, T., & Koyutürk, M. (2011). Comparative analysis of algorithms for next-generation sequencing read alignment. *Bioinformatics, 27*(20), 2790-2796.

Sanger, F., Nicklen, S., & Coulson, A. R. (1977). DNA sequencing with chain-terminating inhibitors. *PNAS, 74*(12), 5463-5467.

Schbath, S., Martin, V., Zytnicki, M., Fayolle, J., Loux, V., & Gibrat, J. F. (2011). *Mapping reads on a genomic sequence: an practical comparative analysis.* Reporte técnico, Statistics for systems biology group, Paris, Francia.

Shang, J., Zhu, F., Vongsangnak, W., Tang, Y., Zhang, W., & Shen, B. (2014). Evaluation and comparison of multiple aligners for next-generation sequencing data analysis. *BioMed Research International, 2014.*

Smith, A. D., Xuan, Z., & Zhang, M. Q. (2008). Using quality scores and longer reads improves accuracy af Solexa read mapping. *BMC Bioinformatics, 9*(128).

Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology, 147*(1), 195-197.

Teubner, J., & Woods, L. (2013). *Data processing on FPGAs.* Morgan & Claypool publishers.

Venter, C., Adams, M. D., Myers, E. W., Li, P. W., Mural, R. J., Sutton, G. G., y otros. (2001). The sequence of the human genome. *SCIENCE, 291*, 1304-1351.

Waidyasooriya, H. M., Hariyama, M., & Kameyama, M. (2013). Implementation of a custom hardware-accelerator for short-read mapping using Burrows-Wheeler Alignment. *35th Annual International conference of the IEEE EMBS* (págs. 651-654). Osaka, Japan: IEEE.

Watson, J., & Crick, F. (1953). Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature, 171*, 737-738.

Zerbino, D. R., & Birney, E. (2008). Velvet: Algorithms for de novo short read assembly using de bruijn graphs. *Genome Research, 18*, 821-829.