Algoritmo sintetizable en hardware para la detección de parásitos bovinos

Guido Ignacio Rombolá [1]

Universidad Nacional de Tres de Febrero, Departamento de Ciencia y Tecnología
grombola@untref.edu.ar

Abstract. El sector agropecuario y agroindustrial es el principal generador de divisas en Argentina. En esta industria, un buen control parasitario permite mitigar grandes pérdidas económicas. Para esto, una de las formas de atacar el problema es a través del uso de fármacos en los animales. El análisis parasitario se efectúa enviando muestras a laboratorios veterinarios especializados, motivo por el cual se demora el tratamiento y se generan costos adicionales tanto para el productor como para el veterinario.
Debido a esto resulta valioso automatizar el conteo de huevos de parásitos en bovinos mediante un dispositivo portátil que se pueda trasladar hasta el lugar en donde habitan estos animales para realizar un diagnóstico in situ.
En este trabajo se propone el desarrollo de un algoritmo sintetizable en hardware (IP Core) con FPGA como plataforma destino para el conteo de huevos de parásitos utilizando síntesis de alto nivel. Los resultados demuestran la factibilidad de implementación, obteniendo un 87% de precisión y un 77% de sensibilidad, y operando a una tasa de 45 y 62 cuadros por segundo para los kits comerciales propuestos: PYNQ-Z1 y ULTRA96V2, respectivamente.

Palabras clave: FPGA, HLS, procesamiento de imágenes.

1 Introducción

1.1 Contexto

Argentina es el segundo exportador de carne vacuna a nivel Sudamericano, quinto a nivel mundial, y es el país con más consumo interno de carne vacuna per cápita [1]. Con un gran esfuerzo, los productores destinan millonarias sumas de dinero a mantener activa la producción y cuidar el rodeo ya que es la materia prima de su negocio.

Las enfermedades endoparasitarias han sido una de las limitantes sanitarias más importantes en la producción de bovinos en sistemas pastoriles de Argentina.

Las parasitosis internas se han convertido en una enfermedad que afecta la salud productiva de los animales, impactando notablemente en la rentabilidad final del

sistema de producción. Algunas consecuencias son la mortandad de animales, pérdida de peso, y disminución de la calidad de la res, entre otras.

La forma de tratar estos problemas en los animales es mediante el uso de los antiparasitarios. Algunos ganaderos tomaron a su cargo el control parasitario, inclinándose por aplicar estos antiparasitarios intensivamente y dejando de lado la actuación profesional. Sin embargo, el control de las infecciones basado exclusivamente en la utilización de antiparasitarios puede generar resistencia de los parásitos a estos productos, conocida como resistencia antihelmíntica. Por eso, es importante hacer un uso racional de los antiparasitarios, con un diagnóstico previo en lugar de hacerlo de manera indiscriminada, ya que se incurren en gastos económicos adicionales sin generar beneficios suficientes. [2]

Idealmente se debe considerar la adopción de un programa que considere tanto los animales como las pasturas, ya que estas últimas alojan la mayoría de los parásitos.

Una de las técnicas de diagnóstico de parásitos se realiza mediante el conteo de huevos que son eliminados por las heces, ya que es un método práctico y de bajo costo. Este método consiste en colocar la materia fecal en una solución sobresaturada de cloruro de sodio para precipitar los huevos hacia la superficie, y realizar el conteo mediante un microscopio en un laboratorio. Es importante realizar un muestreo de aproximadamente el 10% de un lote de animales, porque el conteo de huevos puede tener una gran variabilidad en un mismo lote. [3]

Este conteo tiene algunas limitaciones. Por un lado, puede implicar una subjetividad en los resultados por la fatiga del experto. Además, para llevar a cabo el proceso de diagnóstico y control se invierte tiempo por el traslado de las muestras refrigeradas hasta un laboratorio para analizarlas bajo el microscopio, realizar un conteo manual, y de ser necesario, desparasitar a los animales.

Por estas limitaciones, resulta conveniente la implementación de un dispositivo portátil de detección que permita al veterinario realizar los análisis en el campo. La finalidad de este trabajo es automatizar el conteo de huevos en materia fecal de bovinos.

1.2 **Objetivo**

Para lograr esta automatización, el objetivo es diseñar un algoritmo de procesamiento de imágenes que realice este conteo en base a una imagen de entrada. La plataforma destino es un FPGA, y por eso se dice que es un algoritmo sintetizable en hardware.

Este algoritmo está pensado para ser materializado en un producto pequeño y portable que reciba una imagen capturada por un microscopio portátil USB de bajo costo, y arroje la cantidad de huevos que contiene. El instrumento propuesto permitirá a un profesional veterinario tomar una decisión sin incurrir en tiempos adicionales de traslado y esfuerzo manual de conteo.

1.3 **Límites**

El trabajo se limita al desarrollo del algoritmo que automatiza el conteo recibiendo una imagen de entrada, y se incluyen simulaciones que muestran que es factible

materializar este algoritmo en esa plataforma, contemplando tiempos y uso de recursos.

Queda fuera de los límites de este trabajo la entrega del producto físico de detección portátil, ya que esto implica la materialización del algoritmo en el hardware destino, la obtención de imágenes directamente del microscopio hacia ese hardware, y mostrar los resultados, por ejemplo mediante un display. Para llevar estas tareas a cabo se deben usar herramientas adicionales para integrar el microscopio USB con el algoritmo desarrollado, así también como la salida al display.

1.4 **Marco teórico**

Al revisar la literatura existente se pueden encontrar diversos trabajos de detección de parásitos en animales y humanos, con variadas técnicas y plataformas.

Algunos trabajos utilizan como plataforma de procesamiento una computadora personal tomando imágenes de un microscopio, ya sea utilizando una clasificación por métodos estadísticos seleccionando previamente ciertas características representativas [4], utilizando mecanismos más simples como características que caen dentro de ciertos intervalos [5] o bien utilizando redes neuronales que seleccionan las características automáticamente [6, 7, 8]. Otras soluciones utilizan smartphones en combinación con dispositivos de imagen de fluorescencia [9].

Varias de estas soluciones no son fáciles o convenientes de trasladar ya que suelen involucrar equipamiento especializado y costoso como microscopios de alta calidad, o dispositivos de imagen de fluorescencia para capturar las imágenes. [10] En contraste, la propuesta presentada en este trabajo se elabora para que la solución final sea trasladable, teniendo en cuenta que los costos permanezcan en valores limitados. Con este desarrollo se quiere recibir imágenes tomadas con un microscopio USB de bajo costo, aunque su calidad sea limitada, y procesarlas en una plataforma destino que también sea accesible, pero que permita tener un mecanismo de diagnóstico rápido que optimice tiempos y evite costos de traslado, aunque no se haga una clasificación de los tipos de huevos encontrados. Este tipo de microscopios de bajo costo alcanzan hasta 1000X de Zoom óptico, con foco ajustable, iluminación led con intensidad controlada, y permiten entregar imágenes en resoluciones de 640x480 y 800x600 píxeles. Además, pueden capturar los huevos de parásitos, cuyo tamaño promedio es de 85 micrómetros [11].

1.5 **Justificación de la propuesta utilizada**

Uno de los factores a tener en cuenta es que se busca que el dispositivo que implemente la solución sea portátil. Además, es deseable un reducido consumo de potencia que permita maximizar la autonomía. Algunos enfoques posibles son:

Procesamiento en un servidor en red: enviando la imagen previamente capturada utilizando algún dispositivo con conectividad a internet. Implementando la solución utilizando un software a medida o algún servicio en la nube que permite entrenar un modelo para reconocer los objetos deseados.

Solución de software ejecutándose en computadora portátil utilizando herramientas provistas por los lenguajes de alto nivel, como MATLAB u OpenCV

Computadoras de placa única: como una Raspberry Pi.

Todas las propuestas mencionadas hasta el momento se basan en la recopilación realizada en [12]. Por último, si bien su capacidad de procesamiento es limitada en comparación con las opciones anteriores, es posible considerar un *system-on-a-chip* basado en un microcontrolador. Algunas de sus ventajas son la portabilidad, y su reducido consumo de potencia. Existen algunos enfoques incipientes que utilizan esta plataforma, como el trabajo que realiza Rovai utilizando un kit de desarrollo ESP32 con una cámara para clasificar frutas mediante machine learning [13].

Otra alternativa puede ser mediante un FPGA, que es un chip que puede ser programado para implementar circuitos digitales y ofrece un buen rendimiento ya que permite la paralelización por hardware de algoritmos. Además, se pueden reprogramar, permitiendo flexibilidad en el flujo de diseño. Algunas soluciones son conocidas como *system-on-a-chip*, que incluyen en una misma placa lógica programable (FPGA) más una sistema de procesamiento (CPU).

Estableciendo la comparativa entre un FPGA y un CPU, si una imagen atraviesa una serie de operaciones de procesamiento, el FPGA permite aplicar cada operación sobre un píxel a medida que ese pixel finaliza con la operación anterior. En cambio, el comportamiento secuencial de un CPU hace que hasta que la imagen entera no completó la primera operación, no se pueda pasar a la siguiente, lo que involucra latencias mayores.

En cuanto a las características energéticas, los FPGA se caracterizan por tener un consumo de energía favorable en comparación con un CPU ARM o la GPU [14].

Utilizar la propuesta de procesamiento en servidor en red podría causar que el diagnóstico dependa en gran medida de la conectividad del lugar, y considerando que los animales se sitúan en los campos, esta conectividad podría no ser óptima.

En cuanto a la utilización de una computadora portátil ejecutando un software, implicaría mover un producto de mayor tamaño que en otras soluciones como un FPGA, una computadora de única placa o un microcontrolador. Además, se estaría utilizando un dispositivo de propósito general para una tarea específica. Esto último es relevante ya que, teniendo en cuenta que el dispositivo operaría en condiciones más hostiles (se diagnosticaría en el campo), utilizar y movilizar una computadora portátil en todo momento podría afectar su vida útil. Si se hace en un dispositivo dedicado, podría encapsularse y sellarse debidamente para evitar estos problemas.

Los system-on-a-chip (SoC) basados en microcontroladores no traen estos inconvenientes de portabilidad o el consumo de energía. Sin embargo, no suelen ser la solución ideal en procesamiento de imágenes ya que además de su limitada capacidad de procesamiento, no aprovechan el paralelismo. A pesar de los enfoques incipientes mencionados con esta plataforma, se descarta esta opción.

En ese sentido, quedan las opciones de FPGA o computadoras de placa única. Si bien ambas opciones podrían ser apropiadas, se opta por utilizar FPGA como plataforma destino con la intención de aprovechar el paralelismo, ideal para el procesamiento de imágenes, y por las características de consumo energético favorables descritas.

Se selecciona por estos motivos, para la implementación de la solución, una plataforma basada en FPGA, ya que cumple con los requerimientos de confiabilidad, robustez, portabilidad, velocidad de procesamiento y consumo reducido de potencia.

1.6 Elección de enfoque de desarrollo en FPGA

Para comprender la solución es importante conocer algunos detalles de este hardware y de las herramientas de desarrollo del mismo. Estos conceptos son una recopilación de [15] y [16].

1.6.1 Flujo de diseño en FPGAs

El flujo de diseño para los FPGA se divide en dos etapas: la etapa de frontend y la etapa de backend. La primera tiene que ver con qué herramientas se utilizan para el diseño del algoritmo, y la segunda es el mapeo de las operaciones realizadas al hardware de la FPGA.

Para la etapa de frontend, a la que se limita este trabajo, existen distintos enfoques para generar una descripción de hardware para FPGAs:

- Diseño guiado por modelos y específico de dominio: Se permite la generación de diseños FPGAs utilizando mayormente una interfaz gráfica. Por ejemplo utilizando MATLAB con Simulink.
- Diseño de bajo nivel: utilizando lenguajes de descripción de hardware (HDL por sus siglas en inglés), como Verilog o VHDL. Estos lenguajes dan amplio control sobre el hardware generado. Sin embargo, se requieren mayores conocimientos de hardware.
- Diseño con lenguajes de alto nivel: como C o C++. Al manejar mayores niveles de abstracción, permiten incrementar la productividad. Para esto se utilizan herramientas conocidas como de síntesis de alto nivel (HLS por sus siglas en inglés).

Con cualquiera de las opciones, finalmente se obtiene un diseño a nivel de transferencia de registros (RTL por sus siglas en inglés), que es un nivel de abstracción en donde el estado se almacena en registros y memorias y la lógica entre los registros (por ejemplo, aritmética) es utilizada para generar nuevos estados. En el caso del diseño de bajo nivel, el RTL es escrito por los propios diseñadores de hardware, y en los otros casos, las herramientas generan este RTL.

En particular, se propone para este desarrollo aplicar la metodología de diseño basada en síntesis de alto nivel, utilizando una descripción del diseño de entrada realizada en C++. Las herramientas de síntesis de alto nivel generan una descripción RTL automáticamente a partir de la entrada. De esta manera, se evita tener que realizar un diseño con un lenguaje de descripción de hardware como Verilog o VHDL en donde se trabaja en un nivel de abstracción inferior, debiendo gestionar entre otras cosas, transferencias entre registros.

Además de reducir los costos de desarrollo, HLS permite una exploración rápida del espacio de diseño. Esto quiere decir que aplicando distintas directivas al código fuente de alto nivel se pueden obtener distintas arquitecturas resultantes. En contraste, trabajando con un lenguaje de descripción de hardware, un cambio en la arquitectura implica reescribir la solución, o parte de ella.

Por último, se puede destacar que al escribir código de alto nivel se pueden realizar verificaciones del mismo antes de su transformación a una descripción RTL, que puede tardar desde algunos minutos hasta varias horas. Esto permite mejorar el ciclo

de feedback y reduciendo el tiempo de iteración hasta lograr una solución funcional aceptable. [17]

1.6.2 Herramienta

Entre las herramientas disponibles se optó por Vivado HLS por su popularidad, principalmente en el ámbito académico [18] , que permite encontrar documentación y foros de intercambio de soluciones a problemas que surgen; y por su soporte para procesamiento de imágenes con la biblioteca xfopencv, subconjunto de OpenCV cuyas funciones están optimizadas para su uso en FPGA.

En contrapartida existen algunas desventajas que no se consideraron determinantes para la elección. La herramienta no es open source, aunque el software se puede descargar de manera gratuita desde la web de Xilinx, y las funcionalidades abarcadas en este trabajo no requieren una licencia paga. Además, se genera una dependencia de la tecnología destino ya que aunque se parta de un lenguaje de alto nivel, ciertas dependencias con las que se escribe este código son propias de la tecnología del fabricante. Esto no representa un gran limitante, ya que Xilinx es uno de los fabricantes más populares de FPGA del mercado.

En particular, se utiliza la versión 2019.2 de la herramienta [19]. Ésta provee un entorno de desarrollo que permite realizar las siguientes actividades, entre otras:

- Compilar, ejecutar y depurar un algoritmo.
- Sintetizar el algoritmo en un diseño RTL, con la posibilidad de utilizar directivas para su optimización.
- Generar reportes de uso de área y tiempos.
- Verificar el diseño RTL mediante co-simulación.
- Empaquetar el diseño RTL en un IP Core.

El flujo de trabajo con Vivado HLS parte de código escrito en un lenguaje de alto nivel. En una primera instancia, se codifica el algoritmo y se hacen verificaciones a nivel de software.

En algún punto cuando se quiere verificar si el diseño es acorde al hardware objetivo, se realiza la síntesis de alto nivel en base a directivas que se colocan sobre las distintas funciones o bloques del código y guían el desarrollo de la arquitectura hardware resultante. Esto significa que el uso de diferentes directivas cambian la arquitectura generada, favoreciendo alguna dimensión, ya sea la disminución de las latencias, por ejemplo mediante el paralelismo, o bien la disminución del área, por ejemplo compartiendo recursos. Una directiva que busca optimizar una dimensión podría impactar negativamente en la otra.

Habiendo generado el RTL se puede hacer una co-simulación, que consiste en la comparación de los resultados de la versión software contra los resultados de la versión RTL. Además, se puede evaluar la implementación resultante a través de reportes de ocupación de recursos, en los que se muestran cantidad de registros utilizados, LUTs, memorias, operadores aritméticos y resultados de tiempos. A partir de este punto, las opciones son hacer una exploración del espacio de diseño, iterando sobre la arquitectura resultante utilizando nuevas directivas y comparando resultados, o bien volver sobre el código de alto nivel para reescribir alguna parte del algoritmo.

Finalmente, el diseño RTL se empaqueta en el IP core, apto para su síntesis en FPGA mediante herramientas como Vivado (no confundir con Vivado HLS).

2 Desarrollo

2.1 Descripción de la solución

La solución está desarrollada utilizando un patrón de pipeline en donde se reciben datos de entrada y se realizan operaciones sobre los mismos. Cada operación produce resultados intermedios que son la entrada de la operación siguiente. Consta de tres etapas: preprocesamiento, segmentación y extracción de características, y clasificación. La figura 1 muestra el flujo que recorre una imagen desde que ingresa hasta que se contabilizan los huevos, especificando los datos de entrada y salida de cada etapa.

La imagen de entrada a la primera etapa de la solución se representa como una matriz en donde por cada píxel de la imagen ubicado en la posición (X,Y), siendo Y la fila y X la columna, se almacenan tres valores comprendidos entre 0 y 255. Estos tres valores se corresponden con las intensidades de rojo, verde y azul, y se conoce a esta representación como RGB.

Preprocesamiento

En primer lugar, la etapa convierte la imagen RGB a escala de grises. Esta imagen en escala de grises posee un valor por cada píxel comprendido entre 0 y 255, siendo 0 negro y 255 blanco.

Luego, aplica una binarización de la imagen. Para esto, se elige un valor de umbral U, y se sigue la siguiente regla de asignación:

$$f(x,\ y)\ <=\ U\ entonces\ g(x,y)\ =\ 255$$

$$f(x,y)\ >\ U\ entonces\ g(x,\ y)\ =\ 0$$

Siendo f(x,y) la imagen antes de la transformación, g(x,y) la imagen binarizada, y U el valor de umbral, que se obtuvo empíricamente y en este caso es 116.

Esto da como resultado una imagen binaria, es decir, una imagen con solo dos valores de píxeles posibles, 0 o 255. Con esta imagen binaria se logra descartar el fondo y conservar únicamente los objetos más oscuros para ser identificados en la etapa siguiente (segmentación y extracción de características). El fondo queda representado con el valor 0 (negro) y los objetos a analizarse con el valor 255 (blanco).

Segmentación y extracción de características

Es la encargada de identificar cada uno de los objetos de interés y extraer sus características. Cada objeto de interés, denominado blob, (siglas de binary large object, aplicable por tratarse de una imagen binaria) está compuesto por un grupo de píxeles conectados, tanto en una fila dada como entre múltiples filas.

Para la identificación de blobs, se realizó una implementación basada en [20] y en [21], que es un algoritmo de una única pasada, es decir, con un solo recorrido de la imagen es capaz de enumerar los blobs con sus características. Por eso, se evita el almacenamiento de la imagen completa en memoria, y solo necesita la información de

las últimas dos filas en cada instante de tiempo, permitiendo descartar la información anterior. Evitar el almacenamiento de la imagen completa en memoria permite destinar menos recursos al circuito y que la solución se pueda desplegar en un FPGA de menores prestaciones.

Considerando que en esta etapa se está trabajando con una imagen binaria donde hay dos valores de píxeles posibles, se representa cada fila como una lista de secuencias de píxeles consecutivos de color blanco. De cada secuencia, son relevantes los números de columna de inicio y de fin. El resto de los valores no presentes en esas secuencias se corresponden con píxeles negros. Se conoce a esta representación con el nombre de run-length encoding. Esta representación facilita la comparación de secuencias entre dos filas para saber si están conectadas y forman un blob. Se usará el término "camino" para describir cada una de estas secuencias.

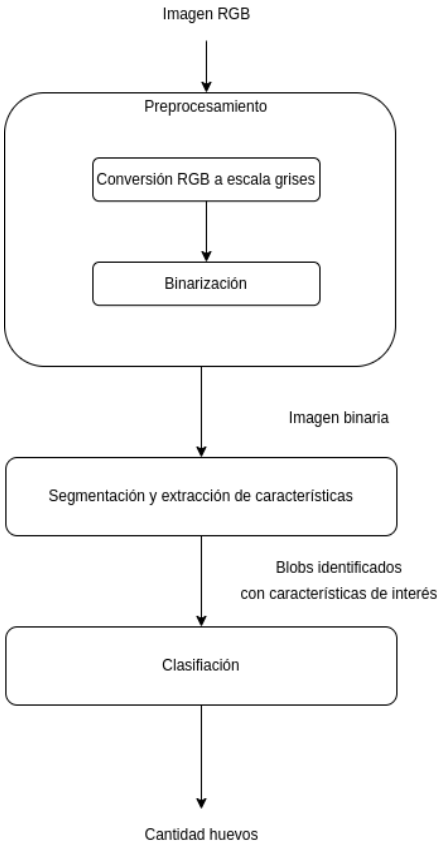


Figura 1: diagrama de la solución

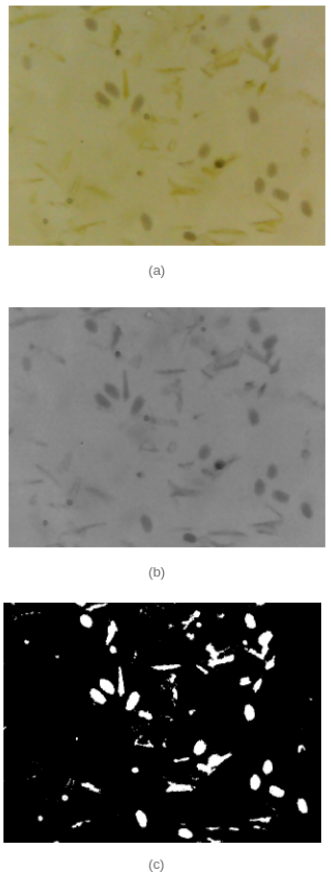


Figura 2: a) Imagen de entrada, b) Imagen convertida a escala de grises, y c) Imagen binarizada

Para identificar los blobs, se deben comparar las filas de acuerdo a las siguientes reglas:

- Si un camino de una fila no se solapa con ningún camino de la fila anterior, se crea un nuevo blob cuyo contenido es únicamente el de este camino.
- Si un camino se solapa con algún camino de la fila anterior, entonces esos dos caminos corresponden al mismo blob. Este ejemplo se ilustra en la figura 3a.
- Si hay dos caminos en una misma fila que inicialmente correspondían a dos blobs, y en la fila siguiente un camino se solapa con los dos de la fila anterior, entonces se combinan los dos blobs ya que son el mismo objeto. Este ejemplo se ilustra en la figura 3b.

Para cada uno de los blobs detectados, se calculan incrementalmente las características a medida que se recorre cada fila. Las características, que se extraen para posteriormente ser utilizadas por la siguiente etapa (clasificación), son:

- *Área*: suma de todos los píxeles de un blob.
- *Mínimo rectángulo que contiene al blob (bounding box)*: rectángulo más pequeño que abarca al blob.
- *Momentos geométricos de orden 1 y 2*: promedios ponderados de las intensidades de los píxeles, que permiten identificar algunos aspectos de la forma de una imagen. [22]

(a) (b)

Figura 3: (a) ejemplo de dos caminos que al solaparse son parte del mismo blob (b) ejemplo de dos caminos independientes que se combinan para formar el mismo blob

Clasificación

Finalmente la información generada por la etapa de segmentación y extracción de características es considerada por la etapa de clasificación, que determina si el blob corresponde a un huevo parasitario o no. Esta etapa realiza una expansión de características a partir de las características extraídas para cada uno de los blobs en la etapa anterior, y se utilizan para realizar la decisión. Estas nuevas características calculadas son excentricidad (grado de elongación de un blob) y relación de área (cociente entre área de la elipse equivalente al blob, y el área del blob).

Si los valores de las características se encuentran contenidos dentro de ciertos intervalos que se obtuvieron empíricamente, entonces el blob es identificado

positivamente. Estos intervalos son: área = [250, 700], excentricidad =[1.5, 6) y rel. área = [1, 1.06].

La salida del algoritmo se corresponde con la cantidad de huevos detectados en la imagen.

Al utilizar el área como característica, las imágenes de entrada deben tener el mismo aumento, que en este caso es 200x.

2.2 **Síntesis del algoritmo**

Las placas de desarrollo propuestas para la solución de conteo de huevos, y por ende utilizadas como objetivo para la síntesis fueron la PYNQ-Z1 y la ULTRA96V2. Éstas cuentan con con un dispositivo system-on-a-chip de Xilinx que combinan lógica programable con un procesador. Son de bajo costo dentro de esta tecnología, y tienen soporte para PYNQ, un framework que permite integrar IP cores utilizando python, y combinarlos con el procesador integrado.

Se utilizaron representaciones de datos de precisión arbitraria para obtener mejores resultados de ocupación de recursos y tiempos de procesamiento. Para aquellas operaciones que requieren de números decimales, se utilizaron representaciones en punto fijo.

Se aplicó además una optimización en la función encargada de incrementar los momentos geométricos para un blob determinado. La ecuación para calcular los momentos geométricos de un blob utilizando la técnica incremental del algoritmo de segmentación se define como:

$$m_{pq} = \sum_{i=0}^{k} \sum_{x=x1[i]}^{x2[i]} x^{p} y_{i}^{q} \tag{1}$$

donde k es el número de caminos (secuencias de píxeles contiguos en blanco) que componen al blob, x1[i] y x2[i] son las coordenadas de inicio y fin de un camino, e yi es el número de fila. Se puede reescribir a la Ecuación (1) como:

$$m_{pq} = \sum_{i=0}^{k} \sum_{x=x1[i]}^{x2[i]} x^{p} y_{i}^{q} = \sum_{i=0}^{k} y_{i}^{q} \sum_{x=x1[i]}^{x2[i]} x^{p} \tag{2}$$

Entonces, considerando las equivalencias de las ecuaciones (3), (4) y (5), los momentos geométricos de orden 1 y 2 pueden ser calculados evitando realizar la suma interna como un bucle. En ese sentido, se reduce la comparación e incremento de la variable iteradora, además la cantidad de operaciones deja de depender de la longitud de cada camino.

$$\sum_{i=1}^{n} i = \frac{n*(n+1)}{2} \tag{3}$$

$$\sum_{i=1}^{n} i^2 = \frac{n*(n+1)*(2n+1)}{6} \quad (4)$$

$$\sum_{i=r}^{n} f(i) = \sum_{i=1}^{n} f(i) - \sum_{i=1}^{r-1} f(i),\ con\ r <= i \quad (5)$$

Adicionalmente, se aplicaron las siguientes directivas de síntesis:

PIPELINE en el loop de la función que verifica si los caminos de una fila se solapan con los de la fila anterior para formar un blob más grande. Permite la ejecución concurrente de operaciones.

INLINE a la función de cálculo de momentos geométricos.

DATAFLOW en la función principal, y en la etapa de preprocesamiento en donde la imagen recibida como un stream de datos pasa por una serie de modificaciones por las distintas funciones de procesamiento. Permite que las funciones se solapen en su ejecución.

LOOP UNROLL en el loop que copia los caminos de la fila actual al arreglo con los caminos de la fila anterior posición a posición, permitiendo ejecutar en paralelo estas copias y disminuyendo la latencia.

Se realizó además una refactorización de código en la función que recorre cada fila en búsqueda de caminos. Originalmente, cada vez que se identificaba un nuevo camino, se comparaba contra todos los caminos de la fila anterior para ver si existía solapamiento para formar un blob más grande.

Luego, esa verificación se trasladó al terminar de escanear la fila completa. En otras palabras, cuando se termina de escanear una fila, se comparan todos los caminos con los caminos de la fila anterior, extrayendo esta verificación a un bucle distinto del que recorre la fila de la imagen.

Si bien la herramienta no lo puede inferir, la verificación del solapamiento no se ejecuta por cada píxel de una fila, sino una vez para cada camino. Como luego del cambio es posible estimar que el nuevo bucle con la verificación se ejecuta menos veces, la latencia que aporta esa verificación es considerada por la herramienta menos veces en el recorrido de la imagen, por ende las estadísticas dan valores más cercanos a lo real y este cambio mejora la latencia de la etapa de segmentación. En consecuencia, esto impacta positivamente en la latencia de la solución general sin perjudicar otros aspectos de la misma.

3 Resultados

3.1 Algoritmo

El algoritmo fue verificado con 30 imágenes obtenidas a partir del uso de un microscopio digital USB compatible con los kits de desarrollo propuestos (PYNQ-Z1 y ULTRA96V2), que contienen un total 234 huevos de parásitos.

Se analizaron los resultados del algoritmo en base a la precisión, definida en la ecuación (6), y sensibilidad definida en la ecuación (7), teniendo en cuenta los resultados obtenidos para:

- Verdaderos Positivos (VP): objetos que son huevos y son detectados como tal.
- Falsos Negativos (FN): objetos que son huevos y no son detectados como tal.
- Falsos Positivos (FP): objetos que no son huevos y son detectados como tal [23].

$$Precisión = \frac{VP}{VP+FP} \quad (6)$$

$$Sensibilidad = \frac{VP}{VP+FN} \quad (7)$$

Los valores de VP, FN y FP fueron obtenidos a partir de una comparación manual de los resultados generados del algoritmo, respecto a los indicados por el experto. Para poder hacer esto, fue necesaria una implementación alternativa de la solución, que, además de indicar la cantidad de huevos detectados, marca las ubicaciones en la imagen. Este proceso se realizó únicamente para la simulación de alto nivel y no se incluye como parte de la solución final apta para hardware debido a que la lógica de marcado implicaría un costo adicional de recursos. En ella, los elementos rodeados por un recuadro azul se corresponden con huevos parasitarios.

Para obtener los resultados de precisión y sensibilidad, se realizó una suma de los VP, FN y FP de cada imagen, para luego aplicar las ecuaciones (6) y (7) sobre los totales. Como resultado, se obtuvo una precisión de 87% y una sensibilidad del 77%.

3.2 **Síntesis**

El core fue sintetizado con una frecuencia de reloj de entrada de 100MHz y co-simulado con la herramienta.

En la Tabla 1 se presentan los resultados de porcentaje de ocupación de recursos obtenidos de las síntesis para ambos kits de desarrollo, demostrando la factibilidad de implementación sobre ambas plataformas.

Tabla 1. Ocupación de recursos (resultados de síntesis)

Kit	***BRAM_18K***	***DSP48E***	***FF***	***LUT***
Pynq Z1	10 %	19 %	11 %	44%
Ultra96V2	6 %	15 %	6 %	41 %

Respecto a los tiempos, se obtuvo una latencia de 22 milisegundos para PYNQ-Z1 y 16 milisegundos para ULTRA96V2, dando una capacidad de procesamiento de 45 y 62 cuadros por segundo respectivamente.

4 Conclusiones

En este trabajo se presentó el desarrollo de un algoritmo sintetizable en hardware (soft IP core) para el conteo automático de huevos de parásitos para ser aplicado en veterinaria ganadera. Los resultados obtenidos demuestran la factibilidad de la

implementación de un sistema portable de bajo costo, capaz de ser utilizado en el campo por especialistas, dada la robustez de la tecnología seleccionada y las ventajas de los FPGAs respecto al bajo consumo. Asimismo, el algoritmo implementado alcanza un nivel de precisión y sensibilidad aceptables (87% y 78% respectivamente) y los resultados de síntesis demuestran la factibilidad de implementación en dos plataformas tanto en tiempo, como utilización de recursos.

Se puede afirmar que el uso de síntesis de alto nivel resultó beneficioso, ya que por un lado permitió evitar tener que profundizar conocimientos sobre un lenguaje de descripción de hardware como VHDL o Verilog. Por otro lado, se optimizaron tiempos de desarrollo por requerir menor cantidad de código necesario para obtener resultados, permitió agilizar la verificación de las soluciones generadas y cambiar la arquitectura de la solución sin un impacto significativo mediante el uso de directivas.

Se puede priorizar como trabajo futuro el desarrollo del dispositivo completo de detección, implementando lo necesario para mostrar resultados mediante algún tipo de display y la obtención de imágenes mediante el microscopio.

Finalmente, se destaca que el presente trabajo tiene dos publicaciones asociadas:

La primera fue aprobada y presentada en las sesiones simultáneas del Congreso Argentino de Sistemas Embebidos 2021 (CASE 2021), realizado de forma virtual, realizado el 1 y 2 de noviembre de 2021, y consiste en una versión intermedia del mismo en el que luego se continuaron haciendo mejoras. Esta versión involucra algunas características adicionales para la clasificación que en la versión final se eliminaron por generar los mismos resultados si no se calculan [24]

La segunda fue publicada en la revista Elektron de la Facultad de Ingeniería de la Universidad de Buenos Aires, en el número semestral Enero-Junio del año 2022, y consiste en una versión actualizada en la que no se utilizaron las características redundantes de la primera publicación [25].

5 Agradecimientos

Al Mg. Lucas Leiva y al Dr. Martín Vázquez, mi director y codirector respectivamente, por el apoyo técnico y la orientación que me brindaron para la redacción del trabajo.

Al Dr. Juan Manuel Toloza, por dar apoyo y establecer el primer contacto con los profesionales veterinarios para discutir esta problemática.

A los doctores Carlos Saumell y Federica Sagües por el interés en la iniciativa y proporcionar el contexto teórico necesario.

A los docentes que tuve durante mi carrera, por brindarme los conocimientos necesarios para ejercer mi profesión.

A mi familia y amigos por apoyarme en el desarrollo de mi carrera.

Referencias

1. G. W. Williams, D. P. Anderson. «The Latin American Livestock Industry: Growth and Challenges». Choices 34.316-2020-617 , 1-11, 2020

2. Lavarello Herbin, A., Schapiro, J. Pérez, R.A., “Uso racional de antiparasitarios, un manejo ecológico en rodeos lecheros de productores familiares del área metropolitana de Buenos Aires, Argentina”, en Congreso Latinoamericano de Agroecología, octubre 2015.

3. Steffan, P. E., Fiel, C. A., Ferreyra, D. A. Diagnóstico de las parasitosis más frecuentes de los rumiantes: técnicas de diagnóstico e interpretación de resultados. Tandil: Abad Benjamin, 2011
4. C. Sommer, «Digital image analysis and identification of eggs from bovine parasitic nematodes», J. Helminthol., vol. 70, n.o 2, pp. 143-151, jun. 1996, doi: 10.1017/S0022149X00015303.
5. K. H. Ghazali, R. S. H. Alsameraai, y Z. Mohamed, «Automated System for Diagnosis Intestinal Parasites by Computerized Image Analysis», Mod. Appl. Sci., vol. 7, n.o 5, p. p98, abr. 2013, doi: 10.5539/mas.v7n5p98.
6. Yoon Seok Yang, Duck Kun Park, Hee Chan Kim, Min-Ho Choi, y Jong-Yil Chai, «Automatic identification of human helminth eggs on microscopic fecal specimens using digital image processing and an artificial neural network», IEEE Trans. Biomed. Eng., vol. 48, n.o 6, pp. 718-730, jun. 2001, doi: 10.1109/10.923789.
7. A. Akintayo, G. L. Tylka, A. K. Singh, B. Ganapathysubramanian, A. Singh, and S. Sarkar, 'A deep learning framework to discern and count microscopic nematode eggs', Sci Rep, vol. 8, no. 1, p. 9145, Dec. 2018, doi: 10.1038/s41598-018-27272-w.
8. Y. Li et al., «A low‐cost, automated parasite diagnostic system via a portable, robotic microscope and deep learning», J. Biophotonics, vol. 12, n.o 9, sep. 2019, doi: 10.1002/jbio.201800410.
9. P. Slusarewicz et al., «Automated parasite faecal egg counting using fluorescence labelling, smartphone image capture and computational image analysis», Int. J. Parasitol., vol. 46, n.o 8, pp. 485-493, jul. 2016, doi: 10.1016/j.ijpara.2016.02.004.
10. "PI-1009 | PrepOneTM Sapphire Blue LED Illuminator with Photo Hood for Smart Devices," Embi Tec. [Online]. Disponible en: https://embitec.com/collections/prepone%E2%84%A2/products/prepone-saphhire-blue-light. [Accedido el 24 de agosto de 2022].
11. "Microscopio Digital Usb 1000x Rgb 8 Led Pc Camara 5mp 1080p, "Mercado libre. [Online]. Disponible en: https://articulo.mercadolibre.com.ar/MLA-886120334-microscopio-digital-usb-1000x-rgb-8-led-pc-camara-5mp-1080p-_JM#position=6&search_layout=stack&type=item&tracking_id=f405c506-86d3-4fa0-b27c-573d2e8c9d12. [Accedido el 24 de agosto de 2022]
12. C. B. Murthy, M. F. Hashmi, N. D. Bokde, y Z. W. Geem, «Investigations of Object Detection in Images/Videos Using Various Deep Learning Techniques and Embedded Platforms—A Comprehensive Review», Applied Sciences, vol. 10, n.o 9, p. 3280, may 2020, doi: 10.3390/app10093280.
13. Rovai, M., 2022. "ESP32-CAM: TinyML Image Classification - Fruits vs Veggies," Hackster. [Online]. Disponible en: https://www.hackster.io/mjrobot/esp32-cam-tinyml-image-classification-fruits-vs-veggies-4ab970. [Accedido el 24 de Agosto de 2022].
14. M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, y P. H. Jones, «Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels», arXiv:1906.11879 [cs, eess], may 2019.
15. D. Koch, F. Hannig, D. Ziener. "FPGA Versus Software Programming: Why, When, and How?" en FPGAs for Software Programmers. Springer Publishing Company, Incorporated, 2016, cap 1, pp. 1-21.
16. J. Cardoso, M. Weinhardt. "High-LevelSynthesis" en FPGAs for Software Programmers. Springer Publishing Company, Incorporated, 2016, cap 2, pp 23-47.
17. R. Millón, E. Frati, y E. Rucci, «A Comparative Study between HLS and HDL on SoC for Image Processing Applications», Elek., vol. 4, n.o 2, pp. 100-106, dic. 2020, doi: 10.37537/rev.elektron.4.2.117.2020.

18. S. Lahti, P. Sjovall, J. Vanne, y T. D. Hamalainen, «Are We There Yet? A Study on the State of High-Level Synthesis», IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 38, n.o 5, pp. 898-911, may 2019, doi: 10.1109/TCAD.2018.2834439.
19. "Vivado Design Suite Tutorial," Xilinx, ene. 2020. [Online]. Disponible en: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug871-vivado-high-level-synthesis-tutorial.pdf [Accedido el 9 de noviembre de 2021].
20. J. Trein, A. T. Schwarzbacher, y B. Hoppe, «FPGA Implementation of a Single Pass Real-Time Blob Analysis Using Run Length Encoding», p. 8, 2008.
21. T. Medina. "Blobs_HLS". [Código fuente]. https://github.com/Tomasmed18/Blobs_HLS/blob/b3195ad1f912f546527de36751fc34dcb942f63e/xf_ip_accel_app.cpp [Accedido el 9 de Noviembre de 2021].
22. W. Burger, M. Burge, "Regions in binary images" en Digital Image processing, 2da edición. Londres, Springer, 2016, cap 10, pp. 209-252.
23. "Classification: Precision and Recall | Machine Learning Crash Course," Google Developers. [Online]. Disponible en: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall?hl=es_419. [Accedido el 9 de noviembre de 2021].
24. G. Rombolá, L. Leiva, M. Vázquez, J. Toloza, "IP Core para control parasitario ganadero", en Congreso Argentino de Sistemas Embebidos, noviembre 2021. [Online]. Disponible en: https://drive.google.com/file/d/14_cYIsp6azGOUhu05HjcPGS-zk7qAvEQ/view
25. G. Rombolá, L. Leiva, M. Vázquez, J. Toloza, F. Sagües, y C. Saumell, «Implementación en FPGA de algoritmo para análisis parasitario», Elek., vol. 6, n.o 1, may 2022, doi: 10.37537/rev.elektron.6.1.149.2022. [Online]. Disponible en : http://elektron.fi.uba.ar/index.php/elektron/article/view/149/281

Anexo

Principios de funcionamiento de un FPGA

Un FPGA es un dispositivo reprogramable. Su operación básica consiste en multiplexores, celdas de memoria de configuración que controlan estos multiplexores, y conexiones entre ellos. Conectando la memoria de configuración a las entradas del del multiplexor, se construye un switch reconfigurable. En la figura 4 se muestra un ejemplo de esto. Si se quiere implementar una compuerta AND, se establece la celda de configuración conectada al input 11 en 1, y el resto en 0. A esto se lo conoce como celda lógica, y cada una de ellas provee una lookup table (LUT) que almacena el resultado de una función booleana. A esto se lo conoce como celda lógica, y cada una de ellas provee una lookup table (LUT) que almacena el resultado de una función booleana. El tamaño de la LUT está dado por el número de entradas, por lo que si se quiere implementar una función con más entradas, se conectan varias celdas entre sí. Las LUTs de las celdas lógicas suelen estar combinadas con flip flops (FF) que se usan para almacenar estados.

Hoy en día los FPGA no proveen solo celdas lógicas, sino una variedad de bloques, como bloques de entrada/salida, multiplicadores, DSPs (bloques para procesamiento de señales que incluyen multiplicadores y sumadores) BRAM (block RAM, utilizadas para almacenar información), ALUs, incluso CPUs completas. A estos bloques se los conoce como hard IP cores (intellectual property cores). En contraste, los circuitos que se pueden implementar como parte de las interconexiones se conocen como soft-IP cores. Entonces, el resultado de esta solución de conteo de huevos es lo que se denomina un soft IP core.

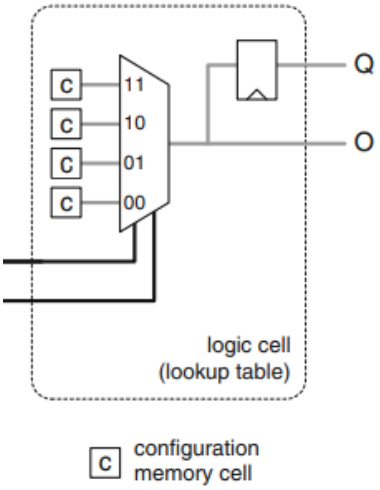


Figura 4: esquema básico de una celda lógica

Cuando se desarrolla un algoritmo en software para una CPU, es importante considerar el tiempo de ejecución y los requerimientos de memoria. En el caso de los FPGAs además de lo anterior, también se debe considerar el dominio espacial, es decir, cuánta área va a ocupar un algoritmo o función lógica. Por ejemplo, para la aritmética de números enteros, el costo para la suma y la resta escala linealmente con el tamaño de los operandos, y es cerca de un LUT por bit. Es común en el diseño de hardware utilizar vectores de precisión arbitraria, y que las operaciones utilicen sólo la cantidad de bits que necesitan.

Directivas de optimización

Inline

Para entender esta directiva, es importante entender que cada función escrita en lenguaje de alto nivel se sintetiza como un bloque RTL. Si existe una función llamada A, y las funciones B y C quieren llamar a A, como se están comunicando con el mismo bloque, tanto B como C comparten el uso del mismo bloque A. Esto se ilustra en la figura 6 (a).

Para el mismo ejemplo descrito, si se usa la directiva inline sobre la función A, se rompe la jerarquía de bloque, por lo que el contenido de A será parte de las funciones B y C, como si A no hubiera existido y su lógica estuviera duplicada. Se representa esto en la figura 6 (b), en donde las funciones B y C pasan a ser B* y C* respectivamente por contener la lógica de A en el interior. Esto puede permitir algunas optimizaciones ya que por un lado, un llamado menos a una función implica menos comunicaciones entre bloques a través de puertos. Además, al estar en el mismo bloque las operaciones de A con B por un lado y de A con C por el otro, la herramienta de síntesis puede aplicar ciertas optimizaciones en caso de que haya alguna operación que se pueda simplificar, que no podría hacerlo si fuera un bloque separado. Como contrapartida, esta duplicación del contenido de A evita compartir el bloque y puede aumentar el área. Es por eso que se suele usar inline para las funciones pequeñas.

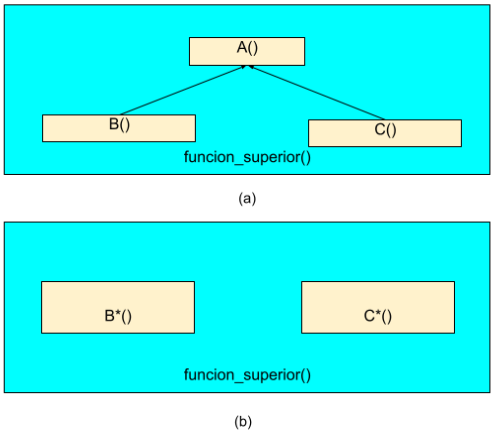
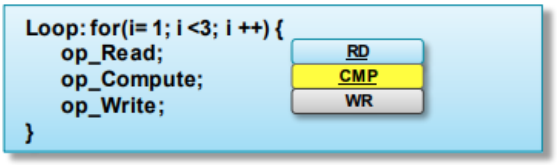


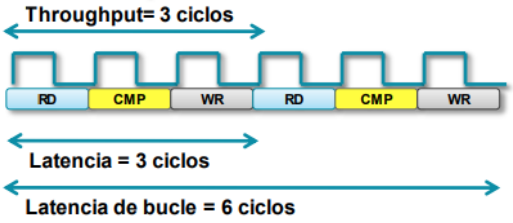
Figura 6: Comportamiento directiva inline. (a) Antes de aplicarla, y (b) después de aplicarla.

Pipeline

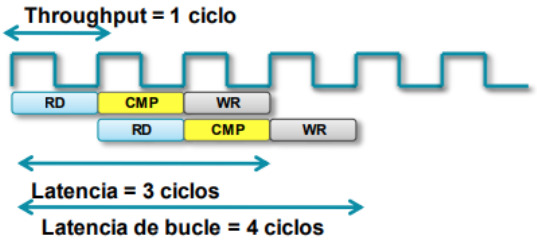
Permite la ejecución concurrente de operaciones. Una función o bucle con pipeline puede procesar nuevas entradas cada N ciclos de reloj, siendo N el throughput. En el caso óptimo, el throughput es 1. Por defecto, al aplicar esta directiva, la herramienta intenta hacerlo con N=1, y de no ser posible porque hay una dependencia de datos entre las operaciones o porque los recursos no están disponibles, se va incrementando hasta llegar a un valor alcanzable. La figura 5 muestra el bucle candidato a ser optimizado, y la comparación entre realizarlo con y sin pipelining. Como se ve, esta optimización permite mejorar la latencia del bucle al permitir solapamiento entre las operaciones del mismo.



(a)



(b)



(c)

Figura 5: (a) bucle candidato a ser optimizado, (b) ejecución original de las operaciones del bucle y (c) ejecución luego de aplicar directiva de pipeline

Dataflow
Es un pipeline a nivel de funciones. Permite que las funciones se solapen en su ejecución. Funciona como el ejemplo del pipeline, pero con funciones en lugar de operaciones dentro de una función.

Loop Unroll (desenrollado de bucles)
En aquellos bucles cuya cantidad de iteraciones está predefinida, desenrollar el bucle significa eliminar la lógica de control para la salida del bucle en caso de desenrollado total, o disminuirla en caso de desenrollado parcial. De ser independientes, desenrollar un bucle permite ejecutar las operaciones en su interior en paralelo.