An FPGA-based Hardware Accelerator for Iris Segmentation

Joe Avey, Phillip Jones, and Joseph Zambreno
Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA 50010
Email: {javey, phjones, zambreno}@iastate.edu

***Abstract*—Biometric authentication is becoming an increasingly prevalent way to identify a person based on unique physical traits such as their fingerprints, face, and iris. The iris stands out particularly among these traits due to its relative invariability with time and high uniqueness. However, iris recognition without special, dedicated tools like near-infrared (NIR) cameras and stationary high-performance computers is a challenge. Solutions have been proposed to target mobile platforms like smart phones and tablets by making use of the Red-Green-Blue (RGB) camera commonly found on those platforms. These solutions tend to be slower than the former due to the reduced performance available in mobile processors. In this paper, we detail a Field Programmable Gate Array (FPGA)-based System on Chip (SoC) approach to help address the mobility and performance challenges that exist in current iris segmentation solutions. Our SoC architecture allows us to run the iris recognition system in software, while accelerating slower parts of the system by using parallel, dedicated hardware modules. Our approach showed a speedup in segmentation of 22× when compared to an x86-64 platform and 468× when compared to an ARMv7 platform.**

***Index Terms*—FPGA, iris recognition, feature extraction, coprocessor.**

I. INTRODUCTION

Biometric authentication uses physical traits of a person to verify their identity, in contrast to using passwords or other keys. The iris is a trait that is relatively stable in-time compared to other physical traits and contains high inter-class variability, making it a good candidate for biometric authentication [1]. Today, iris recognition systems exist to authenticate people for various reasons such as border crossing [2] and welfare programs [3]. However, these systems often take seconds to complete and often require specialized hardware such as near-infrared (NIR) cameras. The convenience and security offered by biometric authentication using the iris is desirable. If authentication of persons via the iris is to become mainstream, it must be supported by fast, mobile systems without the requirement of specialized resources.

Solutions have been proposed to target mobile platforms like smart phones and tablets by making use of the Red-Green-Blue (RGB) camera commonly found on those platforms. These solutions tend to be slower than the former due to the reduced performance available in mobile processors. In this paper, we detail a Field Programmable Gate Array (FPGA)-based System on Chip (SoC) approach to help address the mobility and performance challenges that exist in current iris segmentation solutions. Our SoC architecture allows us to run the iris recognition system in software, while accelerating slower parts of the system by using parallel, dedicated hardware modules. Our approach showed a speedup in segmentation of 22× when compared to an x86-64 platform and 468× when compared to an ARMv7 platform.

The rest of this paper is organized as follows. Section II explains relevant anatomical terms of the eye and the general iris recognition system stages. Section III gives examples of related research in both iris recognition algorithms and the platforms iris recognition has been targeted. Section IV details our software implementation of an iris recognition system and its performance on different processors. Section V describes the hardware architecture that implements the segmentation stage of the iris recognition pipeline, and its performance compared to the software-only implementation described in Section IV. Section VI offers our conclusion and outlines future work.

II. BACKGROUND: IRIS RECOGNITION

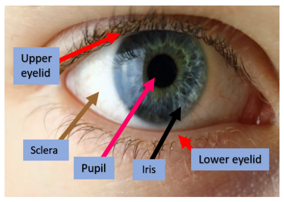
In this section, the relevant anatomical terms and fundamental stages of the iris recognition pipeline are presented. Figure 1 provides a visual overview. The *pupil* is the black circular hole in the center of the eye. It appears a dark, black color because light is absorbed by tissues inside the eye. The *iris* is the colored annulus—ring-shaped object—located between the pupil and the white part of the eye called the *sclera*. The iris can develop in many different colors, but its most interesting characteristic is its texture of arches, furrows, ridges, crypts, and a collarette, all of which make irides highly unique from person to person. The iris develops early on in a human's life and is stable over time [1], which adds to the convenience factor of biometric authentication using the iris.

There are two boundaries that separate the iris from the the rest of the components in the eye. The inner boundary, or *pupillary boundary*, is a circle on the outer edge of the pupil which separates the iris from the pupil. The outer boundary, or *limbic boundary*, is a circle on the outer edge of the iris separating the iris from the sclera, or white part of the eye.

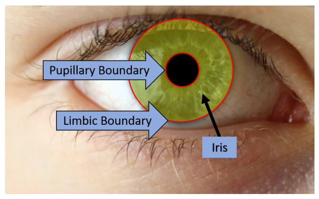
A. Iris Recognition Stages

The iris recognition pipeline is broken into these fundamental stages: *acquisition*, *segmentation*, *normalization*, *feature extraction*, and *classification* (see Fig. 2).

Acquisition. The acquisition stage involves capturing an image of an eye using a camera. In most extant iris recognition systems today, this involves a controlled setting and an NIR



(a)



(b)

Fig. 1: (a) Simple anatomy of the eye. (b) The pupillary and limbic boundaries

imaging sensor. The user is given instructions on where to place themselves and where to look so the system can acquire an image of their eye [4]. The Mobile Iris Challenge Evaluation contests mentioned in Section III aim to solve the problem of restriction to a highly-controlled environment. They target mobile device platforms—which are often equipped with red, blue, and green (RGB) imaging sensors—and base evaluation of their solutions on images of participants taken in random, uncontrolled environments.

After acquisition, images may be post-processed to help with later stages of the pipeline. Post-processing may include methods for enhancing or filtering the image to help with segmentation, or feature extraction. In the case of less restricted iris recognition systems, further methods may be used to localize the eye to aid in segmentation. This allows instructions to be simpler to follow for the user, but may reduce accuracy or runtime performance as a trade-off for convenience.

Segmentation. The segmentation stage involves locating the boundaries of the iris. These boundaries identify the area that contains the iris in the shape of an annulus, or annulus-like shape—in the case of occlusions, non-circular irides, or off-axis gaze. While there are several methods typically used for iris segmentation found in the research literature (e.g. the circular Hough Transform [5], [6], active contours [7], local statistics kurtosis [8], [9], thresholding and binary morphology [8], [9]), in this paper we focus on the integro-differential method, which has well-documented and reasonably high accuracy results.

The integro-differential operator acts as a circle detector after some image pre-processing. The original eye image is first blurred and then we take the partial derivative with respect to increasing radius around a center point. The resulting gradient image highlights the pupillary boundary and limbic boundary due to the change in shade or color between the pupil to iris and the iris to sclera, respectively. The operation then uses an integrator to find the maximum valued circle—with respect to increasing radius from a center point—corresponding to an iris boundary. In research contained in the literature, this operator can be used to find both iris boundaries. This method is applied in many places within the literature and is well described in [1], [10].

Normalization. The normalization stage seeks to perform operations on the image, after segmentation, that make it possible or easier to detect features and remove noise or unwanted data before the feature extraction stage. In iris recognition, common normalization steps include: removing eyelash or eyelid occlusions, removing light reflections, and a polar-coordinate transform to warp the iris annulus into a rectangular image. As is illustrated in Fig. 3, the polar-coordinate transform warps the iris into a rectangular image using the two iris boundaries found in the segmentation step. This unwrapped format makes it easier to extract features from the iris without the need to do polar-coordinate lookup in the feature extraction stage. In the literature, this approach is often called the Rubber Sheet Model in reference to stretching the inner parts of the iris to match the length of the outer boundary to form the unwrapped image. The y-axis of the unwrapped image represents radius and the x-axis represents degrees. Each row in the unwrapped image corresponds to a circle in the iris annulus at a different radius.

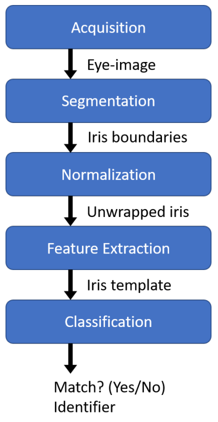


Fig. 2: The iris recognition system model.

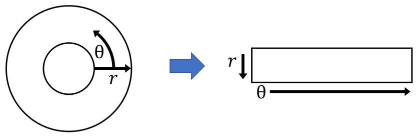


Fig. 3: The polar transform (Rubber Sheet) model.

This stage is also responsible for removing various noise patterns that may appear in the iris that can occur from light reflections and occlusions. Since these patterns tend to fit into well-defined ranges (for example, upper eyelid occlusions most often occur between 45 and 135 degrees in the unwrapped image), an inverse-threshold mask combined with pixel-value histograms can be used to mask out noise [7].

Feature Extraction. The feature extraction stage involves finding a feature vector (often referred to as iris templates in this context) that may uniquely identify the candidate iris. One common approach to building these templates involves wavelet transformations [1], [5], [11]. A simpler alternative is the Ridge-Energy Direction (RED) method originally presented in [9] and further used in [12], [13].

Classification. The classification stage follows feature extraction and seeks to find a match between the local iris template and one or more iris templates previously enrolled in a database. The typical method used to compare two iris templates is the Hamming distance between the iris templates' bit-strings. This metric simply counts the number of differences in the two bit-strings and normalizes the difference to a number between 0.0 and 1.0.

III. RELATED WORK

In 1993, Daugman published algorithms for iris recognition that are still widely used today [10]. Updates to that initial work have focused on performance optimization [1] and on improving the accuracy of segmentation [7].

A. Mobile Iris Recognition

Due to the massive use of mobile devices in today's world, the Mobile Iris CHallenge Evaluation (MICHE) [14] was created to promote biometric authentication, via the iris, on mobile platforms such as smart phones and tablets in less-controlled environments. In many circumstances, the users of these mobile devices access secure data and services that need to be protected. In common cases, these mobile devices possess red, green, and blue (RGB) imaging sensors which MICHE aims to use for convenience and availability. The first MICHE contest was presented in 2015 to promote advances in mobile iris recognition. The contest invited researchers to present their solutions to this problem. An RGB eye-image dataset was created from this contest and is used in this work.

There are several works that use the MICHE dataset to target iris recognition on mobile platforms. In [15], De Marsico et al. layout a modular hybrid approach for a face and iris recognition system. Barra et al. [16] uses a median filter to detect and segment the limbic boundary. This approach is similar to the integro-differential operator approach originally described by Daugman in [1], [10]. The runtime performance of this method—on the order of seconds—suggests improvements would be necessary to build an everyday-use tool. In [17], Abate et al. use a watershed transform, after obtaining an average gradient image across all three RGB color channels, to binarize the original image and then use a circle detector to find both the pupillary and limbic boundaries. In [18]–[20] the authors discuss the results of the MICHE competition, showing that mobile iris recognition is possible but highly performance constrained to the order of seconds for identification.

There are also other works outside of the MICHE competition that attempt to use mobile phones for iris recognition. Kurkovsky et al. was one of the earliest to experiment with iris recognition on the mobile phone [21]. Their work uses the NIR-camera image dataset CASIA and claims consistant performance of under three seconds. In [22], the authors explore performance of extant systems using an RGB-image dataset. The systems tested in this work are meant for NIR-camera images but the authors claim that the systems are mostly successful when using only the red plane of their RGB-images. An experiment such as this could provide interesting insights on how to tune existing systems to make use of mobile phone RGB-cameras.

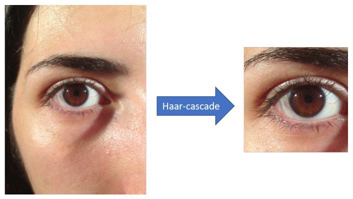


Fig. 4: The original loaded scene and cascade classifier result.

B. Alternative Platform Implementations

There are also several approaches in research that target different platforms than the traditional single-core software-based solutions for iris segmentation, using NIR image datasets. In [23], Raida et al. produce a hardware/software hybrid implementation of iris segmentation using Gaussian smoothing- and morphological-accelerators on a Cyclone-II FPGA. They show a gain in execution-time performance from $\sim$3 seconds to $\sim$2 seconds for a total of 33% speedup when using both types of accelerators. In [24], the authors target multiple platforms to obtain performance trade-offs in building token identifiers for classification/matching. They target a desktop computer, a microprocessor, and an FPGA. In their results, they give performance trade-off analysis when using different resolutions in their FPGA solution. In [25], Hematian presents a heavily pipelined FPGA-based iris localization method. Sakr et al. target an NVIDIA GTX 460 GPU for iris recognition in [26]. In the localization stage, they present a speedup of $10\times$ when compared to a 2.3 gigahertz Intel Core i3 software solution.

IV. SOFTWARE-ONLY IRIS RECOGNITION SYSTEM

This section discusses our iris recognition system software prototype and its results. For this prototype, we make use of the OpenCV library which supports algorithms and data structures that are useful for computer vision applications [27]. The following sections describe the high-level methods employed for each stage.

A. Acquisition

In the first stage of the iris recognition system, an RGB image from the MICHE dataset is read from disk. The scenes in the images of this dataset vary greatly due to their purposefully uncontrolled environment. Because this work does not aim to specifically solve the problem of an uncontrolled environment, the built-in OpenCV cascade classifier method is used to retrieve an eye region of interest (ROI) within the scene. While this classifier is both computationally expensive and necessary to work with the input dataset, it is not considered a fundamental bottleneck in the pipeline as other usage scenarios could require the user to orient their device in a particular fashion. An example acquisition of an image and the cascade classifier result are shown in Fig. 4.

B. Segmentation

In this software prototype, thresholding and morphological operators are used to segment the pupil. In the eye image, the pupil appears as a near-black, circular blob. As the red plane of the image gives the greatest contrast between the pupil and the rest of the eye, it is therefore best suited for pupillary boundary segmentation. A threshold operation is used on the red plane of the eye image to create a pupil mask. The mask may still contain unwanted noise from other dark regions of the eye, such as the eyelashes and eyebrows. To help remove any potential noise and fill in holes from reflections from the camera or environment, two morphological operators—erosion and dilation—are applied to the pupil mask. Finally, we find the connected components in pupil mask. The resulting connected components are filtered based on their circularity and size (i.e. area) with the circularity of a perfect circle being equal to 1.

If there is more than one remaining connected component after filtering, the largest-in-area component is chosen as the pupil. If there are not any remaining components after filtering, then the process restarts from the thresholding operation, with a higher or lower threshold value. An example result of this method for segmenting the pupil is shown in Fig. 5.

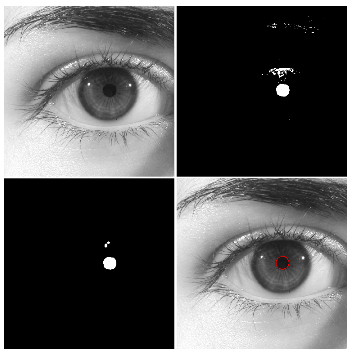


Fig. 5: An example result of pupil segmentation. The original red plane ROI (top-left). The image after thresholding (top-right). The image after morphological operations (bottom-left). The image segmented pupil, identified by a red circle (bottom-right).

1) Limbic Boundary: In this software prototype, the integro-differential method is used to segment the limbic boundary. The eye image must first be preprocessed before attempting to find the limbic boundary. First, a Gaussian blur is used to smooth the image to help get rid of sharp noise. Then, a sobel operator is used to compute the gradient of the eye image. For the standard integro-differential operator the gradient is computed from a center point within the pupil with respect to increasing radius. For simplicity, this gradient is modified to instead compute two gradients—one in the increasing horizontal direction and another in the increasing vertical direction—that are averaged to together. The results of these gradient operators are shown in Fig. 6. One can observe the ring that is created around the limbic boundary. This information is used to segment the boundary.

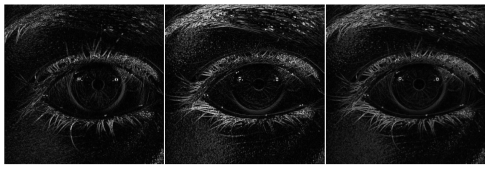


Fig. 6: The gradient images: horizontal gradient (left), vertical gradient (center), and averaged combined (right).

After computing the gradient image, we threshold it with a low threshold value to get a representative binary image. The next step is to find the best-fit circle which corresponds to the limbic boundary. This approach uses a radial accumulator, or integrator, to find the circle containing the highest number of white pixels, corresponding to the limbic boundary.

It is well known that the pupillary and limbic boundaries may not be concentric, i.e. they may not share the same center, even with on-axis gaze. For this reason, the next step in limbic segmentation is an iterative process to find the best center for the limbic boundary. We define a center-search square consisting of points around the pupil center. Each point in the center-search square is tried to find the best-fit limbic boundary. In this work, the center-search square size is arbitrarily chosen as 5x5. In future work, the size of this square can be estimated based on the size of the pupil; however, the runtime of this method is linearly proportional to the size of the square. Each pixel location in the center-search square represents possible centers for the limbic boundary circle. The process at each pixel in the square can be broken down into these two steps:

1) Unwrap the threshold image: As previously discussed, using a modified polar coordinate transform, the threshold image is *unwrapped* to polar coordinates. This process is similar to the Rubber Sheet model defined in [1].
2) Best-fit band: Find the horizontal band with the highest number of white pixels. This band corresponds to the limbic boundary.

The best-fit band step involves finding the horizontal band with the highest number of white pixels in the unwrapped image. A horizontal band in the unwrapped image corresponds to a circle in the original threshold image. This is because the unwrapped image is in a polar space with radius represented by the y-axis and angle represented by the x-axis. Therefore, a perfectly straight band in the unwrapped image corresponds to a perfect circle in the original threshold image. This is the part of the integro-differential operator that acts as a circle detector. We accumulate along circles at different radii. This process is demonstrated in both Cartesian and polar spaces in Fig. 7. The band with the highest number of white pixels—found via accumulation—is considered the best-fit circle for the current center location in the center-search square.

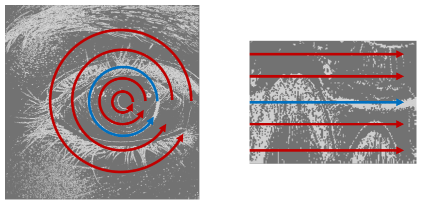


Fig. 7: An example of the best-fit circle operation in both Cartesian (left) and polar (right) spaces. The blue circle (band) corresponds to the best-fit.

In our implementation, the software prototype computes the best-fit circle (band) in the unwrapped image, or polar space, only. It's also worth mentioning that in this work we ignore the accumulation of values between 45 degrees and 135 degrees, and values between 225 degrees and 315 degrees as a static way of removing noise that may be introduced from the upper and lower eyelids, respectively.

C. Normalization

The normalization stage involves creating and formatting an image from the information retrieved in the segmentation stage so that features can be extracted in the next stage. In the software prototype, we first warp the eye image using a polar transform (Rubber Sheet) model. We use the pupillary (inner) and limbic (outer) boundaries discovered in the segmentation stage as input to the polar transform operation. Once the iris has been unwrapped using the approach above, we attempt to remove noise from reflections and eyelid occlusions. One can spot the noise in the unwrapped image. Figure 8 gives labeled examples of noise in this iris image.

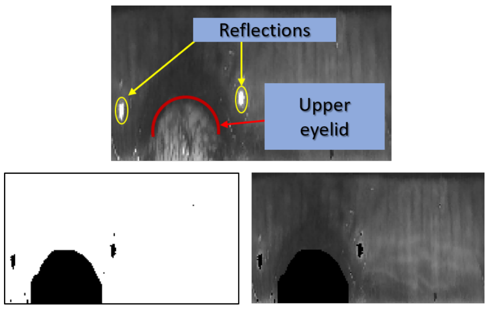


Fig. 8: Labeled examples of noise in the unwrapped iris (top). The noise mask generated from reflection and eyelid detection (bottom-left). The iris after the noise mask is applied (bottom-right).

In this work, we use a white-pixel value to threshold the unwrapped image. This results in a mask that is used to removed the light reflections. Another threshold is used to attempt to detect eyelids. It is not always the case, but often the eyelid is a lighter-shade than the rest of the pupil. We also know that eyelids occlude the iris at 45-135 degrees and 225-315 degrees. This information helps in identifying eyelid occlusions. After thresholding the unwrapped image we can find contours that correspond to eyelid occlusions. If a contour or two contours are found, then a mask is built using the contour information to remove eyelid occlusions. Figure 8 shows a mask built to remove noise from the unwrapped image.

D. Feature Extraction

The Ridge-Energy Direction approach considers the "energy" of the unwrapped image. As explained in [9], energy refers to the magnitude of the ridges that appear in the iris. The direction of the ridges is calculated using two filters, a horizontal and a vertical. First, the unwrapped image is further normalized using local-adaptive histogram equalization. Then, the method builds two separated images using 9x9 convolution filters on the unwrapped image.

Examples of the filtered images are shown in Fig. 9. To finish feature extraction we use the two filtered images to build the result and a mask that signifies which bits in the template are valid to compare in the next stage. The template bit-string and mask are the same size in bits as filtered images are in pixels. At every pixel location in the filtered images, we compare the values. If the horizontally filtered pixel is greater, then we append a '1' to the bit-string, else we append a '0'. This resulting bit identifies the direction of a ridge in the unwrapped iris image. Also, if the greater-valued pixel is less than an energy threshold—defined in this work as the mid-point between a black pixel and a white pixel (128)—then we append a '0' to the mask, otherwise we append a '1' to the mask. A value of '1' in the mask signifies that the corresponding bit in the iris template is valid.

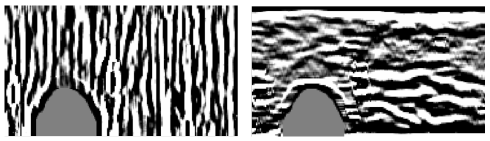


Fig. 9: The horizontally filtered iris (left). The vertically filtered iris (right).

E. Classification

The classification stage seeks to identify a match between the computed local iris template and any number of database templates. For this stage the Hamming Distance metric is used to gauge whether a compelling enough match exists between the local template and a database template. In this work, a Hamming Distance result less than or equal to 0.35 is considered a match. If a match is found, then a unique identifier corresponding to the matched database template is returned. If a match is not found, the user is prompted if they would like to enroll the local template as a new identifier. In a more practical system, enrollment would be controlled by an administrator or other authority.

F. Software Results

This section presents the runtime performance results of the software-prototype iris recognition system. The runtime performance is given for the software-prototype running on two different platforms: x64 (Windows) and ARMv7 (ArchLinux). Table I gives each stage's percentage of the overall runtime for one iteration of the iris recognition pipeline.

TABLE I: Percentage of total runtime for each stage.

Stage	% of Total (x64)	% of Total (ARMv7)
Segmentation	91.9%	98.1%
Normalization	0.0%	0.2%
Feature Extraction	0.0%	1.0%
Classification	8.1%	0.7%

One clear observation is that the segmentation stage is responsible for the majority of the runtime of the software prototype—discounting acquisition—on both platforms. Further profiling (not reported here for sake of brevity) indicated that when split into the sub-stages of pupillary segmentation and limbic segmentation, the two are nearly equal in runtime performance. When broken down further, we noticed that the pupillary segmentation sub-stage involves operations that are commonly found in computer vision systems, such as thresholding and morphological operators. However, the limbic segmentation sub-stage uses methods not commonly found in computer vision systems, such as the modified polar-coordinate transformation and best-fit band computation. Another reason to consider the limbic segmentation sub-stage for hardware acceleration is because it involves several data-independent computations, which could be exploited through parallelism. Each one of these independent computations essentially contains a polar-coordinate transformation followed by a best-fit band operation. For these reasons, a digital hardware design was created to improve performance for the limbic segmentation sub-stage. In a thorough implementation of a high-performance iris recognition system, digital hardware IP such as those contained in Xilinx's HLS Video Library can be used for hardware accelerating the pupillary segmentation sub-stage—thresholding and morphological operations—and the other common computer vision algorithms used in limbic segmentation—Gaussian blur and gradient operations.

V. HARDWARE ACCELERATOR FOR IRIS SEGMENTATION

In this section, we describe a hardware accelerator for the limbic segmentation sub-stage designed to exist in a hardware/software co-implementation of an iris recognition system. We present the runtime performance of the design and compare it to the software prototype's performance when running the same operations. The I/O requirements for the polar-coordinate transformation and segmentation are as follows:

- Inputs
  - The binary-threshold image to be unwrapped
  - The boundary-point arrays which outline the inner and outer boundaries of the area to look for the limbic circle
- Outputs
  - The best-fit band's location (i.e. the radius of the best-fit circle)
  - The best-fit band's total number of ones (so software can compare against other best-fit circles)

In this work, we've normalized the sizes of the polar-coordinate transform images to convenient sizes for designing hardware whilst still maintaing accuracy[1]. For fast read and write access, it would be more convenient to store the images required for this operation in programmable logic memory, such as block RAMs (BRAMs) or in distributed memory, versus loading values individually or in bursts from DDR memory. The binary-threshold image size is 256x256 bits—8 kilobytes—and the unwrapped image size is 128x176 bits—2.75 kilobytes. All dimensions of the images are conveniently normalized to be divisble by eight; allowing easy byte-to-bit conversion, simple and compact storage, and easier write/read logic. Due to the size of the I/O in the polar-coordinate transformation, the decision was made to use block RAMs to store the binary-threshold and unwrapped images. In this work, each image gets their own block RAM to simplify the design. The modified polar-coordinate transformation also requires the boundary-point arrays which specify the annulus area to unwrap. Again for simplification, these arrays are stored in separate block RAMs. Each boundary-point array's size is 176 bytes, one byte for each bit of resolution in the radial dimension (i.e. 176 bytes represent 360 degrees).

Since the decision was made to store the unwrapped image result from the polar-coordinate transform in a block RAM on the programmable logic side of the SoC, it may also be convenient to include the best-fit band operation in the hardware accelerator. This would allow the hardware accelerator to provide a simple result in the form of the size of the best-fit circle's radius and total number of counted ones. Then, the software would only have to compare the results of the best-fit circles calculated by the several instantiated hardware modules to find the one with the highest number of ones—corresponding to the best-fit limbic boundary.

A. *Polar Transform Module*

The polar-coordinate transform calculates each location, (x, y), of the unwrapped image pixels in the binary-threshold image. These locations are loaded from the binary-threshold image and stored in the unwrapped image to be processed by the best-fit band operation after the entire unwrapped image is built.

Because there are multiple reads and read-location calculations, the polar transform functionality is complex enough to warrant a controller. This controller is responsible for transitioning between each set of (r_i, θ_j) locations in the unwrapped image. At each (r, θ), the location of the read-data in the binary-threshold image must be calculated, loaded, and stored in the unwrapped image. The process starts at $(r = 0, \theta = 0)$. The controller increments θ until the

[1]The results obtained from the software prototype also use these size normalizations in order to maintain consistency.

maximum angle is reached at $(r = 0, \theta = 176)$. At this point the controller resets θ and moves to the next row in the unwrapped image at $(r = 1, \theta = 0)$. This process continues until the controller reaches $(r = 128, \theta = 176)$ and the unwrapped image is completed. The architecture diagram for this controller is shown in Fig. 10, with the underlying polar transform module shown in Fig. 11.

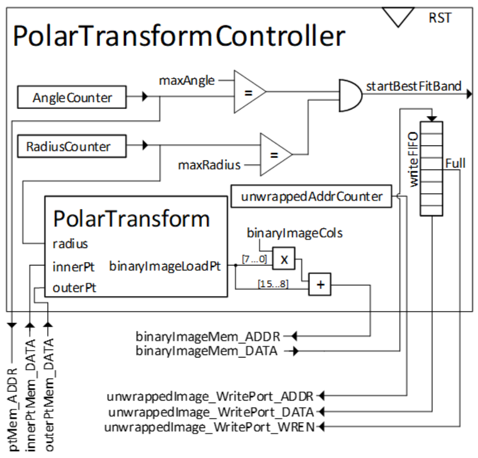


Fig. 10: Polar transform controller Architectural diagram.

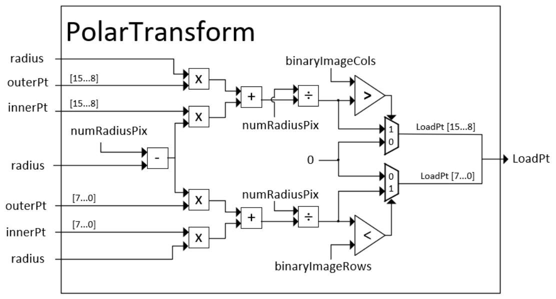


Fig. 11: Architectural diagram for the polar transform module.

B. Best-Fit Band Module

A controller was created for the best-fit band function to deal with the overall operation's complexity. Each band in the unwrapped image is ten rows, each of 176 bits. There are ten bands in total. The goal of this module is to compute these bands in parallel, and after doing so compare their number of counted ones. The band with the highest number of ones is considered the best-fit. The module stores the best-fit band's location (radius) and number of ones in software accessible registers to serve as output. To avoid having massive memory overhead in the band fit controller, we use FIFOs to buffer data from the unwrapped image. There are ten FIFOs, one for each band. Each band in the unwrapped image contains 1,760 bits. For simplicity we make the FIFO depth 176 bits; therefore, we must fill the FIFOs ten times to account for all relevant data in the unwrapped image. We use the controller to manage transfers of data to the FIFOs, and to control when the internal functionality—counting the number of bits that are one—is activated. The internal module reads the FIFO data and counts the number of ones in each band. When all band-data is accounted for, the BestFitBand module compares each band and stores the resulting location and number of ones in output registers. Figure 12 shows the architectural diagram of this controller, and Fig. 13 shows the internal module to calculate the best fit bands.

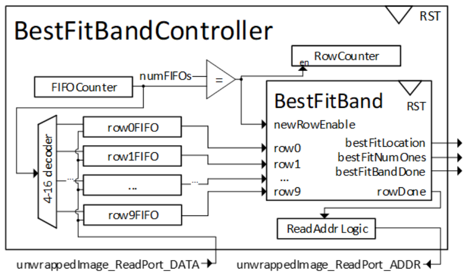


Fig. 12: Architectural diagram for the best fit band controller.

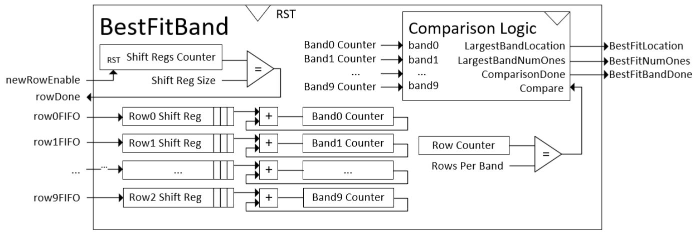


Fig. 13: Architectural digram for the best fit band.

C. Results

The performance results for the iris segmentation IPwere targeted the Xilinx XC7Z020 on the Zedboard . Simulation was used to verify consistency in the results between the software-only prototype and the hardware module. Table II gives the runtime performance of the operations targeted for hardware acceleration.

TABLE II: Runtime performance of operations targeted for hardware acceleration on different platforms.

Internal operation	x64 (ms)	ARMv7 (ms)	Hardware (ms)
Polar transform (x25)	15	326	0.68
Best-fit band (x25)	1	13	0.05
Total	16	339	0.73

The speedup factor of the hardware accelerator was computed using the information from Table II. Overall we observe a 22× speedup when comparing the hardware acceleration method to an x64 platform and a 468× speedup when compared to an ARMv7 platform.

Table III provides resource utilization results for the hardware module on the XC7Z020 platform. We see from these results that we could instantiate twenty-five instances of the iris segmentation module on the target FPGA.

Finally, Table IV attempts to provide a comparison between this paper and other previously published work. While

TABLE III: Resource utilization of the Iris Segmentation module when targeting a Xilinx Zynq-7000 SoC (XC7Z020).

Resource type	Used	Available	Utilization %	Util.×25
Slice LUTs	2053	53200	3.86	96.48
Slice Regs	3878	106400	3.64	91.12
F7 Muxes	49	26600	0.18	4.61
Block RAMs	4	140	2.86	71.43

TABLE IV: Runtime performance summary of related works.

Publication	Stage	Platform Type	Runtime (ms)
[7]	Limbic Seg	CPU (Desktop)	3.5
[1]	Limbic Seg	CPU (Embedded)	90
[16]	Entire Seg	CPU (Embedded)	2000
[17]	Entire Seg	CPU (Embedded)	2000
[28]	Entire Seg	CPU (Embedded)	15000
[25]	Entire Seg	FPGA	6
[29]	Entire Seg	FPGA	25
This work	Limbic Seg	FPGA	0.7
[26]	Entire Seg	GPU	24

these results clearly show the potential of using FPGA-based platforms to accelerate iris recognition, we caution against comparing approaches that use a variety of techniques for segmentation, normalization, and feature extraction, leading to variations in overall accuracy and accuracy/performance tradeoffs.

VI. CONCLUSION

While iris recognition has the potential to become a leading biometric authentication method, to be widely used it must support consumer devices and be quick and convenient to use. In this paper, we have presented a hardware/software design for an iris recognition system, and have analyzed its performance using the MICHE dataset. Our experiments indicate significant speedups over software-only implementations, and illustrate the potential of FPGAs to accelerate this class of application. In the future, we intend on extending this work in an attempt to quantify the accuracy / performance tradeoff, both for our architecture design space and for the various algorithmic options.

REFERENCES

[1] J. Daugman, "How iris recognition works," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 1, pp. 21–30, Jan 2004.

[2] ——, "Iris recognition border-crossing system in the uae," *International Airport Review*, vol. 8, no. 2, 2004.

[3] V. Doshi, "India goes from village to village to compile worlds biggest id database," Jun 2016, [Online]. Available: https://www.theguardian.com.

[4] A. Ross, "Iris recognition: The path forward," *Computer*, vol. 43, no. 2, 2010.

[5] L. Ma, T. Tan, Y. Wang, and D. Zhang, "Efficient iris recognition by characterizing key local variations," *IEEE Transactions on Image processing*, vol. 13, no. 6, pp. 739–750, 2004.

[6] P. Ghosh and M. Rajashekharababu, "Authentication using iris recognition with parallel approach," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 13, no. 5, p. 87, 2013.

[7] J. Daugman, "New methods in iris recognition," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 5, pp. 1167–1175, 2007.

[8] L. R. Kennell, R. W. Ives, and R. M. Gaunt, "Binary morphology and local statistics applied to iris segmentation for recognition," in *Image Processing, 2006 IEEE International Conference on*. IEEE, 2006, pp. 293–296.

[9] R. W. Ives, R. P. Broussard, L. R. Kennell, R. N. Rakvic, and D. M. Etter, "Iris recognition using the ridge energy direction (red) algorithm," in *Signals, Systems and Computers, 2008 42nd Asilomar Conference on*. IEEE, 2008, pp. 1219–1223.

[10] J. G. Daugman, "High confidence visual recognition of persons by a test of statistical independence," *IEEE transactions on pattern analysis and machine intelligence*, vol. 15, no. 11, pp. 1148–1161, 1993.

[11] H. R. Sahebi and S. Askari, "A novel method for iris recognition using bp neural network and parallel computing," *Advances in Computer Science: an International Journal*, vol. 5, no. 2, pp. 1–6, 2016.

[12] C. Liu, B. Petroski, G. Cordone, G. Torres, and S. Schuckers, "Iris matching algorithm on many-core platforms," in *Technologies for Homeland Security (HST), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 1–6.

[13] M. M. Memane and S. R. Ganorkar, "Red algorithm based iris recognition," *genetics*, vol. 1, p. 2, 2012.

[14] M. De Marsico, M. Nappi, D. Riccio, and H. Wechsler, "Mobile iris challenge evaluation (MICHE)-I, biometric iris dataset and protocols," *Pattern Recognition Letters*, vol. 57, pp. 17–23, 2015.

[15] M. De Marsico, C. Galdi, M. Nappi, and D. Riccio, "Firme: Face and iris recognition for mobile engagement," *Image and Vision Computing*, vol. 32, no. 12, pp. 1161–1172, 2014.

[16] S. Barra, A. Casanova, F. Narducci, and S. Ricciardi, "Ubiquitous iris recognition by means of mobile devices," *Pattern Recognition Letters*, vol. 57, pp. 66–73, 2015.

[17] A. F. Abate, M. Frucci, C. Galdi, and D. Riccio, "Bird: Watershed based iris detection for mobile devices," *Pattern Recognition Letters*, vol. 57, pp. 43–51, 2015.

[18] M. Castrillón-Santana, M. De Marsico, M. Nappi, F. Narducci, and H. Proença, "Mobile iris challenge evaluation ii: results from the icpr competition," in *Pattern Recognition (ICPR), 2016 23rd International Conference on*. IEEE, 2016, pp. 149–154.

[19] M. D. Marsico, M. Nappi, and H. Proena, "Results from MICHE II Mobile Iris CHallenge Evaluation II," *Pattern Recognition Letters*, vol. 91, pp. 3 – 10, 2017.

[20] M. De Marsico, M. Nappi, F. Narducci, and H. Proença, "Insights into the results of miche i-mobile iris challenge evaluation," *Pattern Recognition*, vol. 74, pp. 286–304, 2018.

[21] S. Kurkovsky, T. Carpenter, and C. MacDonald, "Experiments with simple iris recognition for mobile phones," in *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*. IEEE, 2010, pp. 1293–1294.

[22] M. Trokielewicz, E. Bartuzi, K. Michowska, A. Andrzejewska, and M. Selegrat, "Exploring the feasibility of iris recognition for visible spectrum iris images obtained using smartphone camera," in *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2015*, vol. 9662. International Society for Optics and Photonics, 2015, p. 96622C.

[23] H. Raida and M. A. YassineAoudni, "Hw\sw implementation of iris recognition algorithm in the fpga," *International Journal of Engineering Science*, vol. 4, 2012.

[24] J. Liu-Jimenez, R. Sanchez-Reillo, and B. Fernandez-Saavedra, "Iris biometrics for embedded systems," *IEEE transactions on very large scale integration (vlsi) systems*, vol. 19, no. 2, pp. 274–282, 2011.

[25] A. Hematian, S. Chuprat, A. A. Manaf, S. Yazdani, and N. Parsazadeh, "Real-time fpga-based human iris recognition embedded system: Zero-delay human iris feature extraction," in *The 9th International Conference on Computing and InformationTechnology (IC2IT2013)*. Springer, 2013, pp. 195–204.

[26] F. Z. Sakr, M. Taher, A. M. Ei-Bialy, and A. M. Wahba, "Accelerating iris recognition algorithms on gpus," in *Biomedical Engineering Conference (CIBEC), 2012 Cairo International*. IEEE, 2012, pp. 73–76.

[27] Itseez, "Opencv library," [Online]. Available: https://opencv.org.

[28] M. Haindl and M. Krupička, "Unsupervised detection of non-iris occlusions," *Pattern Recognition Letters*, vol. 57, pp. 60–65, 2015.

[29] K. Grabowski and A. Napieralski, "Hardware architecture optimized for iris recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 9, pp. 1293–1303, 2011.