Approximate Computing for Biometric Security Systems: A Case Study on Iris Scanning

Soheil Hashemi*, Hokchhay Tann*, Francesco Buttafuoco*† and Sherief Reda*
* School of Engineering, Brown University, Providence, Rhode Island 02912
Email: {soheil_hashemi,hokchhay_tann,sherief_reda}@brown.edu
† Polytechnic University of Turin, Turin, Italy
Email: francesco.buttafuoco@studenti.polito.it

***Abstract*—Exploiting the error resilience of emerging data-rich applications, approximate computing promotes the introduction of small amount of inaccuracy into computing systems to achieve significant reduction in computing resources such as power, design area, runtime or energy. Successful applications for approximate computing have been demonstrated in the areas of machine learning, image processing and computer vision. In this paper we make the case for a new direction for approximate computing in the field of biometric security with a comprehensive case study of iris scanning. We devise an end-to-end flow from an input camera to the final iris encoding that produces sufficiently accurate final results despite relying on intermediate approximate computational steps. Unlike previous methods which evaluated approximate computing techniques on individual algorithms, our flow consists of a complex SW/HW pipeline of four major algorithms that eventually compute the iris encoding from input live camera feeds. In our flow, we identify overall eight approximation knobs at both the algorithmic and hardware levels to trade-off accuracy with runtime. To identify the optimal values for these knobs, we devise a novel design space exploration technique based on reinforcement learning with a recurrent neural network agent. Finally, we fully implement and test our proposed methodologies using both benchmark dataset images and live images from a camera using an FPGA-based SoC. We show that we are able to reduce the runtime of the system by 48× on top of an already HW accelerated design, while meeting industry-standard accuracy requirements for iris scanning systems.**

I. Introduction

Approximate computing is an emerging paradigm that proposes the introduction of insignificant and controlled inaccuracies, such that significant savings can be achieved in design metrics, such as runtime, power, design area and energy. Approximate computing is only suitable for applications where approximate outputs do not degrade the quality of service significantly. Recent growth in machine learning and computer vision applications providing such applications on large scale, has further motivated research in this area [2]. Recent approximate computing methodologies have been proposed both in software [6], [12] and hardware design [9], [14], [7], [5], and from transistor level to cloud computing applications.

We observe that biometric security is another ideal candidate field for the application of approximate computing techniques. Biometric security applications include: finger printing, iris scanners and face identifications. These applications are ideal for two main reasons: (1) the biometrics are data rich, and (2) the difference in biometric signatures of different individuals are large, and as a result signatures from the same individual are considered equivalent even if there are minor differences in them. For instance, the industry standards for iris scanning, e.g., ISO/IEC 19794-6, consider an iris encoding, which is represented by 2048 bits, as high quality even if the quality drops by 25% compared to the ideal case, because there is 1 in 13 billion chance to have a Hamming distance less than 25% between the irises of two different individuals [3].

In this work, we propose an end-to-end biometric security system with an approximate SW/HW pipeline. Using an iris recognition application we showcase how approximate computing methodologies can be effectively implemented on an end-to-end system that is widely deployed. More specifically, our papers makes the following contributions.

- We propose biometric security systems as a novel direction where approximate computing techniques can be readily applied, and we showcase the benefits of such approximations on an iris scanning system, which is a prime method for biometric identification.
- We develop an end-to-end accurate system with approximate SW/HW processing pipelining for iris scanning systems. The complex pipeline consists of four major algorithms, where we identified in total eight knobs to trade-off accuracy of intermediate computations with runtime and energy consumption. We show that while controlled inaccuracies are added in the pipeline, the end encoding outcomes follow the guidelines set in the industry for guaranteed security.
- We propose a novel Recurrent Neural Network (RNN) methodology based on reinforcement learning for Design Space Exploration (DSE) of our SW/HW pipeline flow and identify the best design knobs that minimize runtime subject to accuracy constraints.
- We fully implement our methodology on an FPGA-based SoC using a camera with infrared sensor as input. We evaluate the performance of our system using both standard datasets and live feeds directly from the camera. We demonstrate that significant benefits can be achieved on an accurate end-to-end system using approximate pipeline. We report benefits of up to 48× in runtime while maintaining 100% accuracy on the datasets and live feed.

The rest of the paper is organized as follows. In Section II

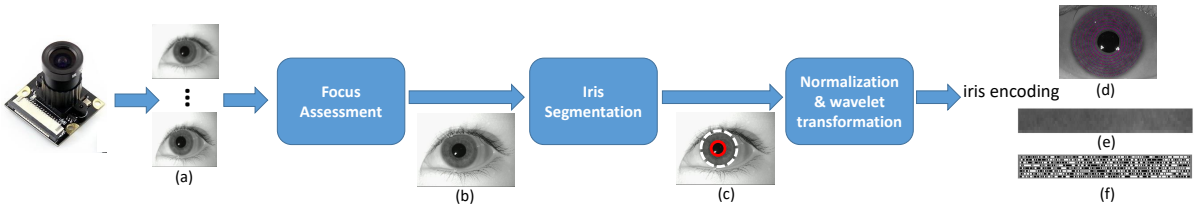

Fig. 1. The components of an iris recognition system.

we provide a background of iris recognition and its pipeline. In Section III, we describe our SW/HW proposed methodology, and our novel design space exploration (DSE) method. We discuss our experimental setup, as well as our experimental results in Section IV-B. Finally, Section V provides the final discussion and concludes the paper.

II. Background

Figure 1 shows the flowchart of commercial iris recognition systems. These systems use a pipeline consisting of four major components that takes input images from a camera and produce as output the 2048 bit encoding of the iris.

1) At the front-end, a *camera* with an infrared sensitive sensor captures multiple frames of an iris illuminated with infrared LEDs as given in Figure 1.a.
2) The *focus assessment* stage assesses the focus of the captured frames and picks the sharpest image for subsequent processing as illustrated in Figure 1.b. The degree of focus for each frame is computed using a convolutional kernel that calculates the energy of the images as described in [3].
3) Next, the iris image is *segmented* and the center points for the iris and the pupil as well as their radii are computed as illustrated in Figure 1.c. Here, solutions based on integrodifferential algorithm [3] and circular hough transform (CHT) [13] have been proposed. In this work, the integrodifferential algorithm is utilized. In integrodifferential algorithm, the whole image is first scanned for candidate pixels, and each candidate pixel is then evaluated using a circular differential methodology while the radius is changed from $Rmin$ to $Rmax$. Next, the candidate pixel with the maximum value is passed to a fine-grain search where a small window around the candidate is evaluated in a similar manner, resulting in the iris coordinates. Finally, and in a similar step, a small window around the iris center point is investigated for best match for pupil.
4) The output of the segmentation algorithm is then fed to the *normalization* algorithm where the iris pixels are subsampled and organized in a Cartesian coordination system. This is achieved by simply spacing the angular resolution and the radial resolution equally, based on the segmentation results. Figure 1.d and 1.e show the subsampling process and the resulting 2-D output of the normalization respectively. Finally, the normalized pixels are *encoded* into 2048 bits using a 2-D Gabor demodulation [3] as shown in Figure 1.f. These 2048 bits form the signature of the iris.

Since the goal is to minimize the response time to the user, the runtime of the system is the prime objective for iris scanning systems.

To highlight the potentials of end-to-end approximate design in dramatically reducing the runtime, we implement a complete SW/HW pipeline of the iris recognition algorithm. Previous works exploring hardware design for iris recognition systems have focused on efficient SW/HW co-design of the algorithm using FPGAs [10], [8]. Our work integrates the use of approximate computing within the SW/HW flow, while performing a novel reinforcement learning based DSE of the SW/HW design knobs to identify their optimal settings.

While many advances have been made in the approximate computing paradigm, most of the work evaluate the quality-energy trade-offs of a single module or algorithm in isolation. Optimal benefits in an end-to-end system can only be explored if the approximate computing techniques utilize all parts of the system pipeline where trade-offs are evaluated in connection with each other. Recently, Raha *et. al.* proposed a full-system approximate design using a smart camera system as a case study [11]. In their system, approximations are introduced using camera resolution scaling, reducing memory refresh rate, and computation skipping. Compared to their work, our work (i) provides a new direction in biometric security systems; (ii) evaluates a far more comprehensive set of trade-offs on multiple algorithms in the pipeline; and (iii) provides a novel reinforcement learning based DSE using an RNN agent. Next, we provide details on our proposed methodologies.

III. Methodology

Our goal is to *minimize the response time* to the user from the time of image capture to the encoding of the iris, under the *constraints* that (1) the encoding is accurate by industry standards, and (2) the resultant SoC can fit within the given resources of our logic device.

We describe next our novel methodologies for approximate computing for iris scanning that achieve our goal. First, in Subsection III-A, we explain our methodology for the partitioning of the pipeline flow between the hardware accelerators and the soft processor. Next, in Subsection III-B, we summarize the approximation knobs that we have identified and explored in this work. Finally, Subsection III-C discusses the methodologies used to explore the design space created by the approximate knobs. Here, we discuss our novel methodology for approximations where an RNN is utilized to find the optimal knob settings. In this section, we also briefly discuss our methodology for computing the runtime and the accuracy of each design point.

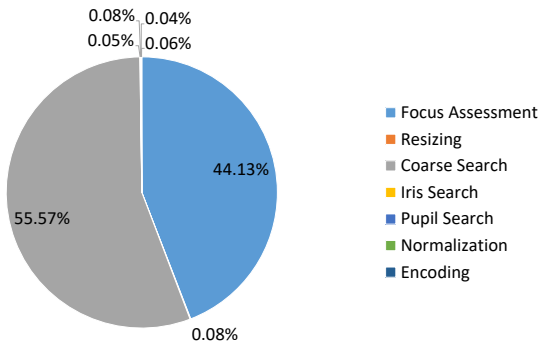

Fig. 2. Percentage of runtime used by various algorithms in the iris scanning pipeline.

A. Proposed SW/HW Partitioning

As an initial step, we profile the pipeline to measure the runtime of its different algorithmic components. To minimize the runtime, we then synthesize the most computational intensive modules into hardware accelerators. The pie chart in Figure 2 summarizes the runtime profiling results when running the flow in software. As shown in the graph, the overwhelming majority of the runtime is spent in the focus assessment and the segmentation components, while the encoding component takes less than 1% of the total runtime. Therefore, we choose to maps these two components into custom hardware accelerators. Here, for the focus assessment, we deploy a full-fledged accelerator with complex buffering to take advantage of data locality. On the other hand, for the iris segmentation, we leave the control sequence in software and move the computationally intensive integrodifferential computation to a hardware accelerator. More specifically, candidate points are located in software and then passed to the accelerator, along with $Rmin$ and $Rmax$, for the computation of the maximum integrodifferential value.

The runtime speedups and the hardware utilization are reported in Section IV-B. Once we determined the accelerators of the system, we explore next the approximation "knobs" of the system, whether in software or hardware.

B. Proposed Approximation Knobs

We propose eight approximation $knobs$ in our SoC iris scanning pipeline, where changing these knobs effectively trades accuracy for runtime benefits. Therefore, we refrain from introducing knobs that do not offer runtime benefits.

- Focus assessment: As the energy of each frame is computed as a convolution of a kernel filter with the image, one obvious accuracy trade-off is the *kernel size* of the filter. Furthermore, instead of computing the focus assessment on entire frames, we can only compute the energy for a subset of the image; i.e., a region of interest (*ROI*), to further reduce the runtime.
- Iris segmentation: We identify six more accuracy knobs in iris segmentation stage. Here, *Npoints* in the number of points used to compute the differential in each circle; *Scale* is the resizing factor used to reduce the segmentation image resolution; *Thresh* represents the threshold beyond which a point is considered to be dark enough to be a candidate; *Rmin* and *Rmax* define the range of radii for which the integration is performed; and *Search Window Size* gives the size of the window around which the local iris and pupil searches are performed.
- Normalization and Encoding: Lastly, in this step, as the design knobs providing accuracy vs. runtime trade-offs also affect the signature specification, in order to stay consistent, we refrain from introducing any knobs.

TABLE I
THE LIST OF APPROXIMATION KNOBS EVALUATED IN THE DESIGN SPACE EXPLORATION. VALUES IN BRACKETS SHOW THE POSSIBLE VALUES.

Pipeline Accelerator	Approximation Knobs [List of values]	Real. in
Focus Assessment	Kernel Size $[8, 6, 4]$	HW
	RoI $[1, 0.78, 0.50, 0.33, 0.20]$	SW
Iris Segmentation	Npoints $[600, 400, 200, 150, 100, 75, 50]$	SW
	Scale $[1, 0.85, 0.75, 0.50, 0.25, 0.20]$	SW
	Thresh $[102, 90, 77, 64, 51, 35, 26]$	SW
	Rmin $[45, 55, 65, 75, 85, 95, 100]$	HW
	Rmax $[180, 170, 160, 150, 140, 130, 120]$	HW
	Search Window Size $[11 \times 11, 7 \times 7, 3 \times 3]$	SW

Table I summarizes the design knobs evaluated in our design space exploration as well as their possible value sets. Here, we also list the component in which each of these knobs are realized based on our SW/HW partitioning. The proposed design knobs result in approximate design space of $648,270$ design corners. Clearly a brute force exploration of the design space in not possible and a design space methodology is required for effective exploration. Section III-C, shows our work in addressing this issue.

C. Design Space Exploration Methodology

Before we can explore the design space to identify the best settings, we need to quantify the accuracy and runtime of different sets of design knobs.

1) Accuracy and Runtime Measurements and Modeling: As evaluating all of the corners on hardware is infeasible, we simulate and formulate the accuracy and the runtime respectively. Since the accuracy performance of one component in the pipeline greatly affects the other components and the final results, we have to estimate the accuracy of a set of knobs using the entire flow through simulation. Thus, to compute the accuracy for each set of design knobs, we run a SW/HW co-simulation of the entire flow. Since such co-simulation can consume significant amount of time, we describe in the experimental results section techniques to speed it up. Unlike accuracy, the runtime of the pipeline flow can be readily decomposed. To that end, we mathematically model the runtime based on the input design knobs and our understanding of the complexity of the algorithm. A summary of our runtime models is shown in Table II. With the runtime formulated, we profiled some training sets of design knobs to quantify the coefficients. We then verified our formulation on another set of knobs settings demonstrating a runtime modeling error of less than 5%. Note that this runtime merely guides the design space exploration and will not translate into inaccuracies.

TABLE II
THE FORMULATION USED TO MODEL THE RUNTIME BEHAVIOR AS A FUNCTION OF THE DESIGN KNOBS. NOTE THAT AS WE DO NOT MODIFY ANY KNOBS IN THE ENCODING COMPONENTS, WE CONSIDER ITS RUNTIME AS A CONSTANT.

Pipeline Component	Runtime Model
Focus Assessment	$\propto RoI^2.KernelSize^2$
Iris Segmentation	$= R_{Coarse} + R_{Iris} + R_{Pupil}$
–Coarse Search	$\propto Scale^3.Thresh.Npoints.(Rmax - Rmin)$
–Iris Local Search	$\propto WindowSize^2.Npoints.(Rmax - Rmin)$
–Pupil Local Search	$\propto WindowSize^2.Npoints.r_{Iris}$
Total	$= R_{FocusAssessment} + R_{Resize} + R_{Segmentation} + R_{Encoding}$

2) Reinforcement Learning Based Design Space Exploration: As a powerful machine learning method, reinforcement learning enables an autonomous agent to make good decisions in its environment through trial-and-error using reward functions. As the agent learns, it starts to tune in on the best set of actions that maximizes its expected reward. Recently this approach has been used to determine the appropriate architectures for general deep neural network classifiers with promising performance [15]. This ability to learn and navigate through complex environment is a perfect fit for our problem. Our multi-objective function, which minimizes runtime and meets encoding error rate requirement, is non-convex. In addition, the input design space contains many dimensions, which makes it hard for traditional exploration methods such as gradient descent. Using reinforcement learning, we direct the agent, an RNN, to learn the best approximations for SW/HW knobs in the exponential design space. Unlike traditional feed-forward neural networks, RNNs have the ability to produce arbitrary-length output sequence. In this case, we encode SW/HW knobs to form a "sentence" [15]. The RNN is then used to predict the best sentence to optimize the systems metrics. During the learning process, the agent seeks to change its predicted sentence to maximize its reward function.

The words in the predicted sentence corresponds to approximations for the knobs. Using an accuracy co-simulation flow and runtime models, we then evaluate the impact of the approximations as shown in Figure 3. Our proposed reinforcement learning methodology efficiently explores the

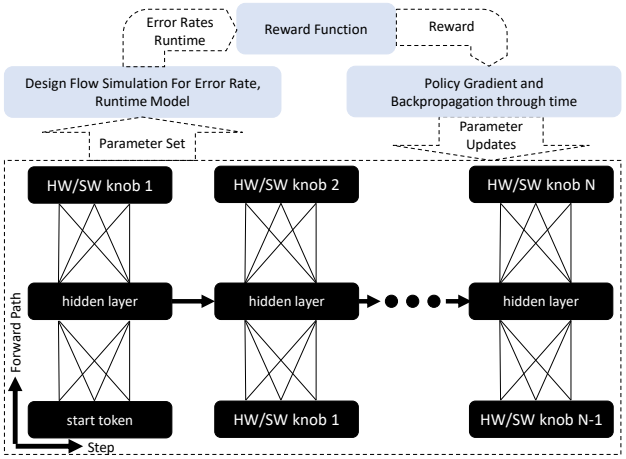

Fig. 3. Design Space Exploration with Reinforcement Learning using RNN.

Algorithm 1: Reinforcement Learning based DSE

```
// Design_Knobs: List of tunable design knobs
   with their possible approximations.
// Approx_Knobs: The approximations for design
   knobs.
Input : Design_Knobs
Output: Approx_Knobs
Initialize(RNN);
for iteration = 1 to N do
    In = START_TOKEN;
    for step = 1 to Number of Design Knobs do
        probs = forward RNN with In as input;
        In = convert probs to one-hot encoding;
        // get design knobs choices
        knobs[step] = argmax(probs);
    end
    rt_current = GetRuntime(knobs);
    err_rate = SimulateErr(knobs);
    avg_err = Average(err_rate);
    max_err = Max(err_rate);
    reward = GetReward(rt_current,rt_best,avg_err, max_err);
    grad = backpropagate(RNN, reward);
    // Update RNN parameters using the gradients
    Update(RNN, grad);
    if max_err < threshold AND runtime < best_runtime then
        rt_best = rt_current;
        Approx_Knobs = knobs;
    end
end
return Approx_Knobs
```

solution space by trying to maximize a multi-objective reward function R, which is defined as:

$$R = \begin{cases} -1, & \text{if } max\_err > 0.35 \\ P, & \text{elif } rt_{current} < rt_{best} \\ -0.1, & Otherwise \end{cases}$$

where rt is the runtime and max_err is the maximum encoding error rate among all the test cases, and $P \in [0, 1]$ is a positive reward for when the candidate sample improves the overall runtime. When a design does not meet error rate constraints, we impose a negative reward.

We employ a two-layer RNN with 30 neurons in the hidden layer. Algorithm 1 describes of design knobs sampling and RNN training process. First, the network weights are initialized according to Xavier method [4]. The activation functions for the hidden layer is $\tanh$, and the output is fed through a softmax layer. Then for each time step, the output is converted to a one-hot vector and fed as input into the network in the next time step. Once the knobs values are chosen, we compute the runtime using our model and error rates through simulation. The reward signal is then determined by the runtime improvement and error rate of the bit encoding. Next, we discuss the details of our learning process.

We train the RNN agent using softmax policy gradient, which aims at maximizing the expected reward:

$$J(\theta) = E_{P(Arch_{1\to K};\theta)}[R] = \sum_{n=1}^{K} R \cdot P(Arch_n | Arch_{n-1}; \theta)$$

where θ is our RNN parameters, R is the reward signal, and $Arch_{1\to K}$ is the values for all the K knobs in the design.

The gradient of $J(\theta)$ can be computed as:

$$\nabla_\theta J(\theta) = \sum_{n=1}^{K} E\left[R \cdot \nabla_\theta \log P(Arch_n | Arch_{n-1}; \theta)\right].$$

Using this computed gradient, we update our RNN parameters using the Root Mean Square Propagation optimizer.

While RNN can efficiently explores the breadth of the design space, it can takes significant amount of time to zoom in on local optimal point. To further improve the runtime benefits, we propose to perform a local search (LS) step using the best result obtained from the RNN. Here in one iteration, we change each parameter one step as long as it does not violate the accuracy.

IV. Experimental Results

In this section we evaluate our methodology empirically, considering both runtime and accuracy performance. We compare our proposed methodology against a greedy and a local search based heuristics (similar approach to methodology proposed in recent work [11]) and report its performance.

A. Experimental Setup

For our experiments we use a Spartan6 Xilinx development board interfaced to a 5 MP Videology camera with infrared optical filter and infrared LEDs for illumination. This 24B5.05USB3 camera features a unique 10 bit digital output port, which allows a direct interface to the raw image sensor pixel data. We also use an Agilent 34410A multimeter to monitor the current and measure power consumption accordingly. Figure 4 shows our camera and FPGA setup. We compile and synthesize all our results on the FPGA board and confirm correct functionality. While we test and run all our designs on the FPGA; for DSE, we co-simulate the accuracy of the SW/HW system and model the runtime. To speed up the computationally demanding co-simulation, we use Verilator [1] to compile the Verilog-based hardware accelerators into C-based simulators, and then use gcc to compile all the components in software. For runtime, we use the methodology described in Section III based on training samples of runtime from the actual board. To validate our performance and to compare against industry standards, we use two sources: (1) images from MMU open source dataset, and (2) live feeds captured from our camera system. Since MMU provides still images, we bypass the focus assessment module for their evaluation. For images captured from camera, we explore the complete flow where the energy of 10 frames are evaluated before the sharpest image is passed to the rest of the pipeline.

In order to assess accuracy, we cross validate the signature of each image in the dataset, using the approximate knob settings, against all of the signatures of the same eye in the repository when computed with full quality. To ensure 100% accuracy in the design, if at any point the maximum hamming distance error of any two images from the same eye goes above 0.35% the design is discarded. Using this threshold ensures a false positive rate of 1 in 133,000 [3]. Next section provides our results.

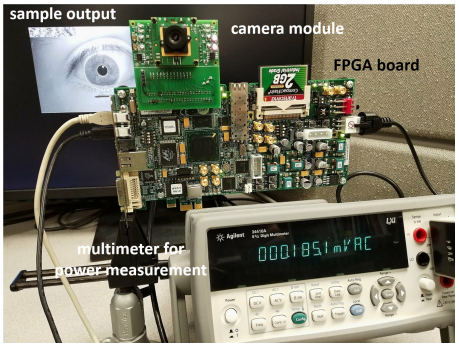

Fig. 4. Our camera and FPGA board Setup.

TABLE III
The components chosen for hardware acceleration, the corresponding speedups, and the hardware utilization of each accelerator.

Pipeline Component	Speedup	HW utilization (%)
Focus Assessment	1234×	25.71
Iris Segmentation	6.8×	13.24
System		15.42

B. Results and Discussion

We first evaluate the benefits achieved from mapping part of the computing pipeline into custom hardware accelerators. As discussed in Section III-A and as commonly practiced, we map the algorithms that have the highest contribution to the total runtime of the iris scanning pipeline to hardware. Thus, we manually translate and map the focus assessment and the segmentation components into hardware accelerators, and leave the remaining parts of the flow to run as software on the MicroBlaze processor. After mapping these two components, the total logic utilization of our device reaches about 60%. Table III shows the speedups when focus assessment and segmentation are mapped to hardware accelerators, together with the logic utilization. These results are merely the benefits from hardware acceleration and use no approximations.

Next, we evaluate the performance of our proposed DSE methodology. Figure 5 shows the design points evaluate using our proposed RNN methodology. Here, the baseline point shows the average error of the SW/HW design without approximation. Using our reinforcement learning based DSE method, our design can achieve up to $42\times$ speedup while still maintaining the standard accuracy limits. Using the proposed RNN+LS method, the algorithm achieves $48\times$ in speedup. We stress that these speedups are due to approximate SW/HW processing and they are on top of the savings achieve due to the HW acceleration.

We also compare our proposed method against a greedy and a local search heuristics. In the greedy approach, starting from the beginning of the pipeline we approximate each design knob as much as possible before moving to the next one. The process is continued until all knobs have been visited. On the other hand, for the local search approach, we iteratively change each quality knob individually by one in order to generate a new design point. The design point with the highest *gradient*, where the gradient is defined by the ratio of runtime

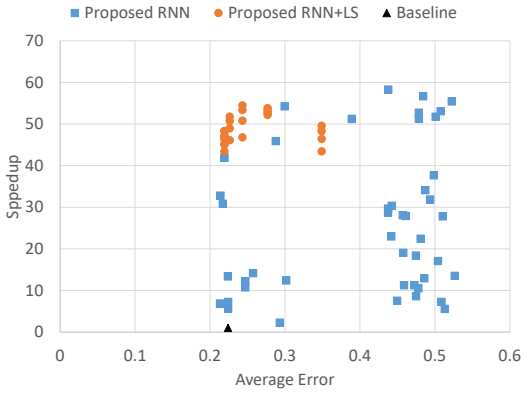

Fig. 5. The design space explored using the proposed RNN, as well as RNN+LS.

TABLE IV
THE COMPARISON OF THE RESULTS USING THE PROPOSED METHOD AD COMPARED AGAINST PURE GRADIENT DESCENT AND GREEDY METHODS.

Design	Speedup	Ave. Error (%)	Design Points Evaluated
RNN+LS	48×	21.92	63
Greedy	37×	21.44	21
Local Search	42×	21.38	132

improvement over quality degradation, is chosen as the parent for the next iteration. Table IV summarizes the trade-offs in speedups and effort offered by each design space methodology. Further, as shown in the table, the proposed method results in the highest speedup while requiring significantly less effort when compared to the gradient descent. On the other hand comparing to the greedy, higher speedups are provided. Note that here, the number of design points evaluated reported in the table directly indicates the efficiency of the DSE algorithm.

Finally, we evaluate the output design of the our methodology on the board to verify the runtime and the accuracy performance. We also compare the result of our methodology with pure software and software/hardware codesign methods. While for the previous experiments, we relied on the MMU benchmark dataset, for this experiment we use our camera system to capture iris images and run the complete flow. Here, we highlight the immense benefits of exploring the hardware/software codesign domain in conjugation with the approximate design knobs exploration. Table V summarizes the results. As shown in the table, significant benefits are achieved in terms of both runtime, and the total energy. Compared to a pure SW implementation, our approximate SW/HW pipeline is able to achieve a speedup of 378× while meeting the industry standard accuracy requirements.

V. CONCLUSIONS

In this paper we identified biometric security as a potential application domain for approximate computing, and we illustrated this potential through a comprehensive case study on iris scanning system. We devised a SW/HW flow that processes input images from a live camera to produce the final encoding of the iris. Our pipeline flow consists of four major algorithms, where we identified eight design knobs that can trade-off design metrics with accuracy as measured by the Hamming distance of the final iris encoding to the golden encoding. We devised a novel recurrent neural network methodology that uses reinforcement learning for design space exploration. Using a comprehensive SoC implementation in a FPGA-based system that receives inputs from a camera sensor with infrared optics, we showed that we can minimize runtime, which is the main objective of iris scanning systems, by 48× compared to exact SW/HW while fitting in the given FPGA resources and meeting the target accuracies of the iris encoding as set by industry standards.

TABLE V
THE HARDWARE CHARACTERISTICS OF THE END-TO-END SYSTEM.

Design	Runtime (s)	HW (%) Utilization	Energy (kJ)	Memory (MB)
Pure SW	2419.6	15.42	47.90	0.69
SW/HW	198.3	54.37	3.94	2.20
Approx. SW/HW	6.4	46.42	0.12	0.89

Acknowledgments: The authors would like to thank L. Camacho and A. Viola from Videology Engineering for the helpful discussion, and Prof. R. Iris Bahar for early discussions on the project related to its SW/HW co-design aspects. This work is partially supported by a RICC grant and NSF grant 1420864.

REFERENCES

[1] Verilator, the fastest free verilog hdl simulator [online], url = https://www.veripool.org/wiki/verilator.
[2] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar. Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency. In *Design Automation Conference*, 2010.
[3] J. Daugman. How iris recognition works. In *IEEE Transactions on Circuits and Systems for Video Technology*, 2004.
[4] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *International Conference on Artificial Intelligence and Statistics*, 2010.
[5] S. Hashemi, R. I. Bahar, and S. Reda. Drum: A dynamic range unbiased multiplier for approximate applications. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 418–425, 2015.
[6] M. Imani, A. Rahimi, and T. S. Rosing. Resistive configurable associative memory for approximate computing. In *Design, Automation Test in Europe*, 2016.
[7] S. Lee, L. K. John, and A. Gerstlauer. High-level synthesis of approximate hardware under joint precision and voltage scaling. In *DATE*, pages 187–192, 2017.
[8] M. Lopez, J. Daugman, and E. Canto. Hardware-software co-design of an iris recognition algorithm. 2011.
[9] K. Nepal, S. Hashemi, H. Tann, R. I. Bahar, and S. Reda. Automated high-level generation of low-power approximate computing circuits. *IEEE Transactions on Emerging Topics in Computing*, PP(99):1–1, 2017.
[10] H. Ngo, J. Shafer, R. Ives, R. Rakvic, and R. Broussard. Real time iris segmentation on fpga. In *International Conference on Application-Specific Systems, Architectures and Processors*, 2012.
[11] A. Raha and V. Raghunathan. Towards full-system energy-accuracy tradeoffs: A case study of an approximate smart camera system. In *Design Automation Conference*, pages 74:1–74:6. ACM, 2017.
[12] A. Sampson, J. Nelson, K. Strauss, and L. Ceze. Approximate storage in solid-state memories. *ACM Trans. Comput. Syst.*, 32(3):9:1–9:23, 2014.
[13] Q.-C. Tian, Q. Pan, Y.-M. Cheng, and Q.-X. Gao. Fast algorithm and application of hough transform in iris segmentation. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*, volume 7, pages 3977–3980 vol.7, Aug 2004.
[14] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. Salsa: Systematic logic synthesis of approximate circuits. In *Design Automation Conference*, pages 796–801, 2012.
[15] B. Zoph and Q. Le. Neural architecture search with reinforcement learning. *International Conference on Learning Representations*, 2017.