S-QRD-ELM: Scalable QR-Decomposition-Based Extreme Learning Machine Engine Supporting Online Class-Incremental Learning for ECG-Based User Identification

Yi-Ta Chen, Student Member, IEEE, Yu-Chuan Chuang, Student Member, IEEE, Li-Sheng Chang, and An-Yeu Wu, Fellow, IEEE

Abstract—User identification enables secure access to data and machines in smart factories. Compared with other modalities, ECG-based user identification is rising due to its intrinsic liveness proof and invulnerability to spoofing without contact. On the other hand, as new employees are registered at the factory, the ECG-based user identification system needs to be updated based on the new coming data. This scenario can be defined as an online class-incremental learning (O-CIL) problem. By exploiting hardware-software co-design, this work presents a Scalable QR-decomposition-based extreme learning machine (S-QRD-ELM) engine that can effectively and efficiently support O-CIL for ECG-based user identification. At the software level, we apply the concept of “the others” class and inversion-free QR-decomposition (QRD) recursive least squares to the S-QRD-ELM. This makes S-QRD-ELM achieve 79.7% higher accuracy in the O-CIL scenario compared with the neural network trained with back-propagation (BP-NN). At the hardware level, a one-dimensionally-diagonally-mapped linear array (1D-DMLA) is proposed to efficiently compute the QRD and back-substitution (BS) operations inside the S-QRD-ELM, reducing 98.5% of the silicon area. Moreover, the integrated processing element (PE) design with the unified COordinate Rotation DigiDigital Computer (u-CORDIC) further reduces 15.3% of the silicon area. The chip achieves 0.02µJ/sample and 2.47µJ/sample inferencing and learning energy efficiency, respectively, which is 6.4× and 28.5× than the state-of-the-art. To the best of our knowledge, the proposed highly-energy-efficient S-QRD-ELM engine is the first chip to meet the requirements of O-CIL for ECG-based user identification.

Index Terms—ECG user identification, online learning, class incremental learning, extreme learning machine, QR-decomposition, CORDIC.

I. INTRODUCTION

SMART devices in the Industrial Internet of Things (IIoT) generate massive amounts of data during monitoring machine status. Secure access to these IIoT data and machines becomes crucial, and user identification comes in handy [1]. Compared with traditional identification methods, such as identification (ID) cards and human-generated passwords, biometric identification has become popular due to its unique biological characteristics. Among all biometrics, electrocardiogram (ECG) signals have emerged as an attractive biometric modality due to their two key advantages: 1) ECG signals originate from the electrical activity of the heart, thus providing intrinsic liveness proof, and 2) ECG-based user identification system is invulnerable to spoof since the ECG signals cannot be easily eavesdropped without physical contact [2]. In addition, ECG signals are still available from wearable devices even in cleanrooms or COVID-19-protected hospitals, while other biometrics (e.g., iris, fingerprint, and face ID) become unavailable due to isolations of cleanroom garments and protective coveralls [3].

An existing ECG-based user identification system [2] is shown in Fig. 1. The ECG data is collected with wearable devices equipped with ECG sensors. Then, the collected ECG data is processed, and the predicted user ID is outputted with a pre-trained neural network (NN) model in the smart ECG processor. However, new authorized users may be registered at the user identification system. It requires the entire system to collect the new subject’s data and retrain the identification system on the cloud with all the data again, which is quite inefficient.

Instead of requiring all the data again, the classification model in the user identification system should incrementally learn the newly registered users without “forgetting” the previously learned ones. This scenario is called “online class incremental learning (O-CIL),” which is the most difficult task in continual learning due to the lack of task IDs and class IDs during inferencing [4]. O-CIL involves both “online learning (OL)” and “class incremental learning (CIL).” However, training back-propagation-based NN (BP-NN) with only new subjects’ data easily induces “catastrophic forgetting (CF),” which causes severe accuracy
degradation due to the recency learning bias towards new classes [5].

To avoid the CF issue, a scalable extreme learning machine (S-ELM) [6] is proposed with a recursive least squares (RLS) solution provided by the online sequential ELM (OS-ELM) [7]. The concept of “the others” class is proposed in S-ELM to better initialize the new class’s parameters. Moreover, they have proven that the weights learned with S-ELM are as same as that of OS-ELM, knowing the total number of classes in advance (including new classes). However, although S-ELM can effectively handle O-CIL, the matrix inversion operations during its learning process are computation-intensive and unsuitable for direct hardware implementations.

In this paper, we present a Scalable QR-decomposition ELM (S-QRD-ELM) engine that can support: 1) Inferencing, 2) OL, and 3) O-CIL, for ECG-based user identification, as shown in Fig. 2. The main contributions of this paper are as follows:

1) **Develop a Hardware-Friendly S-QRD-ELM Algorithm:** We Propose a Hardware-Friendly S-QRD-ELM Algorithm to Alleviate the Inverse Matrix Operation in the Original S-ELM Algorithm. Instead of the RLS Solution Applied in S-ELM, S-QRD-ELM Updates the ELM With the QRD-RLS, an Inversion-Free Solution. “The Others” Class Concept Is Also Combined Into the S-QRD-ELM to Achieve O-CIL Capability. From the Experimental Results, S-QRD-ELM Achieves the Same Accuracy as S-ELM in Both OL and O-CIL Scenarios While Avoiding Matrix Inversion Operations. Compared With the BP-NN, S-QRD-ELM Achieves 79.7% Higher Accuracy in the O-CIL Scenario for ECG-Based User Identification.

2) **Design a VLSI Architecture for S-QRD-ELM:** An S-QRD-ELM Architecture Is Presented With Several Hardware Optimizations. First, a One-Dimensional Diagonally-Mapped Linear Array (1D-DMLA) Architecture Is Proposed to Efficiently Compute the QRD and Back Substitution (BS) Operations. Compared With the 2D Directly-Mapped Systolic Array (SA), Our Proposed 1D-DMLA Architecture Reduces the Area by 98.5%. Second, by Utilizing Hardware-Sharing Techniques, We Propose an Integrated Processing Element (PE) Design With a Unified COordinate Rotation DIgital Computer (u-CORDIC) Circuit Design. Compared to the Non-Integrated PE Design, Our Integrated One Reduces 15.3% of the Area and 22.4% of the Power Consumption. Finally, a CIL Module Is Proposed to Support O-CIL With Only 4.7% Area and 4.95% Power Overhead.

The S-QRD-ELM is fabricated with the TSMC 40nm technology. This chip achieves 0.02μJ/sample and 2.47μJ/sample inferencing and learning energy efficiency, respectively. Compared with the state-of-the-art inferencing-only ELM chip [8], the S-QRD-ELM engine outperforms it by 6.4× inferencing energy efficiency. Compared with the state-of-the-art FPGA-based OS-ELM accelerator with OL capability [9], the S-QRD-ELM engine outperforms it by 28.5× learning energy efficiency. To the best of our knowledge, the proposed highly energy-efficient S-QRD-ELM engine is the first chip to meet O-CIL requirements for ECG-based user identification.

II. PRELIMINARIES

A. Extreme Learning Machine (ELM) [10]

Extreme Learning Machine (ELM) is a classifier based on a single-layer feedforward network (SLFN). During the training of ELM, the training dataset $\mathbb{N}$ can be denoted as $\{(x_i, t_i)\}_{i=1}^{N}$, where $x_i \in \mathbb{R}^{1 \times D}$ is the input data vector, $t_i \in \mathbb{R}^{1 \times M}$ is the target vector for each input data, and $N$ is the size of the input dataset. Input data $X$ and output target $T$ are the matrix form of input data $[x_1, x_2, \ldots, x_N]$ and output target
During the initialization process, we need to assign the network’s input weight matrix \( a \) and bias matrix \( b \) from random sampling. The optimization process of the ELM is to find the optimal output weight \( \hat{\beta} \) such that the difference between output target \( T \) and the output of the ELM \( HH \) has the smallest value. The hidden-layer output \( H \) can be calculated as follows:

\[
H = g(a \cdot X + b),
\]

where \( g \) is the activation function. \( \hat{\beta} \) can be solved by calculating the multiplication of the pseudo-inversion of the hidden layer output \( H \) and the target data \( T \) based on the least mean square (LMS) solution,

\[
\hat{\beta} = \arg\min_\beta ||H \cdot \beta - T|| = H^T T = (H^T H)^{-1} H^T T. \tag{2}
\]

**B. Online Sequential Extreme Learning Machine (OS-ELM) [7]**

ELM requires collecting all the input data and output targets at once to generate the \( \hat{\beta} \), which only fits into applications without online updating or perfect separation between the training and inferencing phases. Online Sequential ELM (OS-ELM) [7] has been proposed to solve the \( \hat{\beta} \) with the recursive least squares (RLS) algorithm. Assumed the initial input data is \( \mathbf{X}_0 = \{(x_i, t_i)\}_{i=1}^{N_0} \), where \( N_0 \) is the size of the initial data, and an incoming chunk at time \( t \) can be denoted as \( \mathbf{X}_t = \{(x_i, t_i)\}_{i=1}^{N_t} \), where \( N_t \) is the size of the chunk at time \( i \). The training phase of the OS-ELM can be separated into two steps:

**Step1: Initialization**

Given an initial batch of input data \( \mathbf{X}_0 \), and the random input weight matrix \( a \) and bias matrix \( b \). The initialization of the output weight can be denoted as

\[
\beta_0 = P_0 H_0^T T_0, \tag{3}
\]

where

\[
P_0 = (H_0^T H_0)^{-1}. \tag{4}
\]

\( H_0 \) is the initial output of the hidden layer, which can be obtained using Eq. (1) with the initial batch of input data.

**Step2: Online Learning**

In the online learning phase, given a new arrival of input data \( \mathbf{X}_{t+1} \) in time \( t+1 \), the output weight can be updated as

\[
\beta_{t+1} = \beta_t + P_{t+1} H_{t+1}^T (T_{t+1} - H_{t+1} \beta_t), \tag{5}
\]

where

\[
P_{t+1} = P_t - P_t H_{t+1}^T (I + H_{t+1} P_t H_{t+1}^T)^{-1} H_{t+1} P_t, \tag{6}
\]

and \( H_{t+1} \) is the output matrix of the hidden layer, which can be obtained using Eq. (1) with \( (t+1)th \) chunk of input data.

**C. Scalable Extreme Learning Machine (S-ELM) [6]**

Although OS-ELM can update the model in the OL scenario, it still needs to know the total number of classes before the training process, which is not the case in O-CIL. Therefore, S-ELM [6] is proposed to make OS-ELM capable of learning in the O-CIL scenario. The S-ELM model possesses an additional output class called “the others.” This class serves as a reference for the output weight when it encounters unseen classes during the training phase. Because there is no data belonging to “the others” class, the output weights of “the others” class tend to make the S-ELM not predict the presented data as “the others” class. When the unseen class data is presented in the input data, we initialize the output weight of the unseen class by copying the “the others” class’s output weights. S-ELM not only makes new classes inherit the knowledge from the previous data but also simplifies the modification of the output weight matrix when unseen classes are presented. It is also proven that the \( \beta \) trained by S-ELM is the same as that of OS-ELM knowing the total number of classes in advance. Compared with Progressive ELM [11], which randomly initialized the new class’s weight, S-ELM can maintain its accuracy better during the O-CIL process.

Assumed that there are \( M \) neurons in the hidden layer and the initial dataset \( \mathbf{X}_0 = (X_0, T_0) \), the steps of the S-ELM training algorithm are as follows:

**Step1: Initialization**

First, we modify the initial batch data from \( (X_0, T_0) \) to \( (X_0, T_0)_{0 \times (C_0+1)} \), where \( N_0 \) is the initial batch size and \( C_0 \) is the class number of the initial data, by appending \([-1]_{N_0 \times 1} \) to \( T_0 \) which serves as the target vector of “the others” class. The input weight matrix \( a \) and bias matrix \( b \) are initialized as same as OS-ELM. We then calculate the hidden layer output matrix \( H_0 \) with \( X_0 \) by Eq. (1). Finally, given that there are \( C_0 \) classes in \( N_0 \), we calculate the initial value of \( P_0 \) and \( [\beta_0]_{M \times (C_0+1)} \) with equations (3) and (4).

**Step2: Online Class Incremental Learning**

In the O-CIL process, we first examine whether there are unseen class data in the upcoming batch \( \mathbf{X}_{t+1} \). Assumed that there exists \( c \) new class(es) in the data at time \( t \), the output weight would be modified as follows:

\[
[\beta_t]_{M \times (C_t+1+c)} = [\beta_t]_{M \times (C_t+1)} [\Delta \beta_t]_{M \times c}. \tag{7}
\]

where

\[
[\Delta \beta_t]_{M \times c} = [\beta_t]_{\text{theothers}} \cdots [\beta_t]_{\text{theothers}}. \tag{8}
\]

It should be noted that \( [\beta_t]_{\text{theothers}} \) is the last column of \( [\beta_t]_{M \times (C_t+1)} \) in S-ELM. If there is no new class data in \( N_{t+1} \), we can preserve the original \( \beta_t \). Afterward, we modify the current batch data from \( (X_{t+1}, T_{t+1}) \) to \( (X_{t+1}, T_{t+1})_{N_{t+1} \times (C_{t+1}+c)} \) by appending a \([-1]_{N_{t+1} \times 1} \) to \( T_{t+1} \). Finally, we can calculate \( \beta_{t+1}, P_{t+1} \) with equations (5) and (6).

Although S-ELM enables the CIL capability of ELM, it contains matrix inversions, which has high computational complexity of \( O(n^3) \). Therefore, we propose the hardware-friendly and inversion-free Scalable QR-decomposition-based
ELM (S-QRD-ELM) algorithm based on the Given rotations with efficient VLSI architectural designs in the following section. We also summarized the related works and the proposed S-QRD-ELM in Table I.

III. PROPOSED SCALABLE QR-DECOMPOSITION-BASED EXTREME LEARNING MACHINE (S-QRD-ELM) ALGORITHM

In this section, we will elaborate the S-QRD-ELM algorithm for both OL and O-CIL. The evaluation of the S-QRD-ELM algorithm will also be introduced.

A. S-QRD-ELM With OL Capability

During the learning process of ELMs, the final goal is to solve the $\beta$ matrix. Most works solve it either with Eq. (2) or equations (3)-(6). However, as the matrix inversion operation has the computational complexity of $O(n^3)$, direct implementations following these equations are inefficient. Therefore, we propose another idea to solve the $\beta$ matrix with the QRD-RLS algorithm [12], which is commonly used in adaptive filtering.

Back to the process of solving $\beta$ using Eq. (2), we focus on the first equation:

$$\hat{\beta} = \arg\min_{\beta} ||H\beta - T||,$$

where $H \in \mathbb{R}^{N \times M}$, $N > M$, is of rank $M$. Because the Euclidean norm is invariant under unitary transformations, we can rewrite the Eq. (9) to

$$\hat{\beta} = \arg\min_{\beta} ||Q^{T}(H\beta - T)||,$$

where $Q \in \mathbb{R}^{N \times N}$ is unitary and $Q^{T}H = R \in \mathbb{R}^{N \times M}$ is upper triangular. The calculation of the matrix $Q$ and $R$ is by the QR-decomposition. The solution to Eq. (10) is

$$Q^{T}H\beta = Q^{T}T.$$  \hspace{1cm} (11)

The LHS of Eq. (11) can be written as

$$Q^{T}H\beta = R\beta = \begin{bmatrix} R_u \\ 0 \end{bmatrix} \beta,$$

where $R_u \in \mathbb{R}^{M \times M}$ is a full rank matrix. Then, the RHS of Eq. (11) can be written as:

$$Q^{T}T = \begin{bmatrix} T_u \\ T_l \end{bmatrix},$$  \hspace{1cm} (13)

where $T_u \in \mathbb{R}^{M \times C}$ and $T_l \in \mathbb{R}^{(N-M) \times C}$. We further simplified Eq. (11) as:

$$R_u \beta = T_u.$$ \hspace{1cm} (14)

For better notations, the $R_u$ matrix will be simplified as $R$ and the $T_u$ matrix will be simplified as $u$ in the following article. Note that Eq. (14) describes a triangular system of equations, which can be solved by back substitution (BS).

With the above derivation, we propose the learning and inferencing process of the S-QRD-ELM algorithm for the OL scenario:

1) **Learning Process:** We Utilize the Random Projection of the Original ELM to Project the Raw Data $X$ to the Hidden Layer Matrix $H$, and Obtain the Output Target Matrix $T$, Where $X \in \mathbb{R}^{N \times D}$ Is With the Size of Input Training Data $N$ and Input Dimension $D$ of the ELM Model, $H \in \mathbb{R}^{N \times M}$ Is With the Size of Input Training Data $N$ and the Number of Hidden Nodes $M$, and $T \in \mathbb{R}^{N \times C}$ Is With the Size of Input Training Data $N$ and the Number of Output Nodes $C$. Afterward, the Elements of Matrices $H$ and $T$ Are Inputted Into the QRD Phase Sequentially. After QRD Computations, the Matrices $H$ and $T$ Are Transformed Into the Matrices $R$ and $u$, Respectively.

2) **Inferencing Process:** At the Start of the Inferencing Phase, We Need to Solve the Output Weight $\beta$ Matrix Using BS With the Matrices $R$ and $u$ Obtained From QRD. For Every Testing Data $x \in \mathbb{R}^{M}$, We First Obtain the Hidden Matrix From Eq. (1) and Multiply It With the $\beta$ Matrix to Get the Predicted Output Vector. Finally, We Find the Argument That Gives the Maximum Value From the Predicted Output Vector and Return It as the Predicted Label.

To obtain a valid output weight matrix $\beta$ during the inferencing process, the input data number accumulated from the previous training process must be larger than the hidden node number. Moreover, the $H$ matrix obtained in the previous training process must also have a rank equal to the hidden node number.

B. S-QRD-ELM With O-CIL Capability

We apply the concept of “the others” class from S-ELM to enable the O-CIL capability of S-QRD-ELM. Fig. 3 depicts the concept of “the others” class and the procedure when facing others’ class. The concept of “the others” class and the procedure when facing others’ class.

We modify the training procedure of the base S-QRD-ELM algorithm to apply “the others” class concept and matrix extension process. First, we need to append the $[-1]_{N \times 1}$
matrix to every target matrix during the training process, where \( N \) is the number of the input training data. Assume that there are \( M \) hidden nodes and \( C_t \) learned classes at time \( t \) in the S-QRD-ELM model. The size of the current \( u \) matrix is \( M \times (C_t + 1) \), where the last column stores the information of “the others” class. At the beginning of the training process, we first examine whether the unseen class data has appeared in the upcoming batch \( S_{t+1} \). If there exist \( c \) new classes in the upcoming data, the \( u \) matrix of the S-QRD-ELM model will be extended as follow:

\[
[u_t]_{M \times (C_t + 1)} = [[u_t]_{M \times (C_t + 1)} [\Delta u_t]_{M \times c}] \quad (15)
\]

where

\[
[\Delta u_t]_{M \times c} = [[u_t]_{\text{heathers}} \cdots [u_t]_{\text{heathers}}], \quad (16)
\]

\( [u_t]_{\text{heathers}} \) is defined as the last column of \( u_t \) in the S-QRD-ELM model. If there is no unseen class data in the batch \( S_{t+1} \), we can preserve the original \( u_t \). Finally, with the hidden layer matrix \( H_{t+1} \) and the one-hot-encoded target matrix \( T_{t+1} \), the matrix \( R \) and modified matrix \( u_t \) can be updated with the original training process.

The flow chart of the S-QRD-ELM and its comparison with the RLS-based S-ELM algorithm is summarized in Fig. 4. Compared with the S-ELM algorithm, the S-QRD-ELM algorithm has several advantages. First, S-QRD-ELM does not require an initialization phase; that is, no initialization data are required, and the entire learning process can be achieved by the QRD. Second, S-QRD-ELM is an inversion-free algorithm, which can be further accelerated with efficient hardware design. Third, the matrix \( \beta \) can be solved on-demand at the start of the inferencing process. In contrast, the S-ELM needs to solve matrix \( \beta \) after receiving each batch of data, which reduces its energy efficiency. In summary, the S-QRD-ELM algorithm is more hardware-friendly and suitable for hardware implementations.

C. Evaluation of the S-QRD-ELM Algorithm

To evaluate the effectiveness of the S-QRD-ELM algorithm for ECG-based user identification, we utilize the ECG samples in the MIT-BIH Normal Sinus Rhythm Database (MIT-BIH NSRDB) [13]. The raw ECG signals in the dataset have a sampling rate of 128Hz. Before we input the raw ECG signal to the ELM classifier, several preprocessing steps are utilized. First, the R-peaks of the ECG signal are detected with the Pan-Tompkins algorithm [14]. After R-peak detection, we segment the ECG signal based on the position of the R-peak. Among the ECG signal from one heartbeat, we select the QRS segment as the target because it is less sensitive to physical and emotional variations, which can further reduce the intra-class variability [15], [16]. Thus, we only select a 0.5-s window to extract the QRS segment, and the central point among the 0.5-s window segment is the R-peak. Due to the sampling rate of the ECG signal being 128Hz, 64 data points are segmented, which contains 32 data points after the R-peak, the R-peak, and 31 data points before the R-peak in our settings. Finally, the ECG segments are normalized where the mean is equal to zero, and the standard deviation is equal to one. We selected ten subjects from a total of 18 subjects of the MIT-BIH NSRDB to do further experiments to match the specification of the S-QRD-ELM engine in section II. For every subject, 256 segments are selected, where 128 segments are the training data and the remaining 128 segments are the testing data. Two sets of experiments will be conducted. We first compare S-QRD-ELM with S-ELM in both OL and O-CIL scenarios. After that, the comparison between S-QRD-ELM and BP-NN will be introduced. All the parameter settings of these models are shown in Table II.

1) Comparison Between S-ELM and S-QRD-ELM: We first compare the S-QRD-ELM and the S-ELM in the OL scenario. The training data are shuffled and split into ten batches to update both ELMs in an online-sequential manner. We examine the accuracy of both ELMs after learning every batch of data with a total of 1280 testing data. The experimental result is shown in Fig. 5. From the experimental result, we can
see that the line from the S-QRD-ELM exactly matched that of the S-ELM. This means that the learning effectiveness of S-QRD-ELM is the same as the S-ELM with different learning algorithms. The final accuracy reached by the S-ELM and S-QRD-ELM is 91.71%.

Next, we compare the accuracy between the S-ELM and the S-QRD-ELM engine in the O-CIL scenario. In this scenario, the training data of the ten subjects are arranged in order and inputted into both ELMs. After every subject’s data are inputted into the ELMs, the corresponding testing data are used to obtain the current accuracy. For example, in the first round, user 1’s training data is inputted, and user 1’s testing data is used to obtain accuracy. In the second round, user 2’s training data is inputted, followed by the testing data of both user 1 and user 2 are applied to obtain the accuracy of the second round. Therefore, we can obtain ten accuracy values and compare them between both ELMs. The experimental result is shown in Fig. 6. From the experimental result, we can see that the line from the S-QRD-ELM matches that of the S-ELM. The final accuracy of both models after learning ten subjects is 91.41%. This experiment shows that S-QRD-ELM has the same O-CIL capability as S-ELM, which has the corresponding mathematical proof.

2) Comparison Between S-QRD-ELM and BP-NN: The settings in this experiment are the same as the O-CIL setting of the previous paragraph. The BP-NN is an SFLN trained with backpropagation with hyperparameters shown in Table II. To be fair to both models, we set the hidden node number of both models equal to 128. The experimental results are shown in Fig. 7. Although BP-NN has better learning capability than S-QRD-ELM because its parameters in the first layer can be finetuned by the BP algorithm, the effect of CF highly hurts its accuracy in the O-CIL scenario. It shows that the S-QRD-ELM outperforms BP-NN by 79.7% accuracy in the O-CIL scenario for ECG-based user identification. Therefore, we can conclude that when facing the CF issue in the O-CIL scenario, the best method is to replace the training algorithm with an iterative solution such as BP to an analytic solution such as S-QRD-ELM.

IV. PROPOSED S-QRD-ELM ENGINE

In this section, we present the design detail of the S-QRD-ELM engine supporting O-CIL. Fig. 8 shows the overall block diagram of the S-QRD-ELM engine. The system architecture, finite state machine, and architecture mapping of both training and inferencing processes are described in the following subsections.

A. System Architecture

The S-QRD-ELM engine primarily consists of a hidden matrix calculator, a target matrix generator, a system controller, three on-chip memories, a global buffer, a CIL module, a SIMD inference engine, and a 1D diagonally-mapped linear array (1D-DMLA). The hidden layer matrix calculator generates the hidden layer matrix with random projections on the input data. In our design, we apply linear feedback shift
FIGURE 8. Block diagram of the proposed S-QRD-ELM engine.

Fig. 8. Block diagram of the proposed S-QRD-ELM engine.

registers (LFSRs) to generate the pseudo-random sequences with \([-1, 1]\) for the random input weight \(a\) and bias \(b\).

To avoid poor accuracy due to projecting the input data into the same or similar space, we utilize 64 LFSRs with different seeds to generate the weights for the corresponding 64 dimensions of the input data. One single LFSR is used to generate the bias. The multiplication between input data and input weights can be implemented with multiplexers. Finally, the output of the hidden matrix calculator, which is the hidden matrix \(H\), can be accumulated with the adder tree followed by a rectified linear unit (ReLU) activation function.

The target matrix generator transforms the inputted label into the one-hot encoded vector in the target matrix \(T\) for the learning process. As only one data is inputted into the S-QRD-ELM for each timestamp, we can utilize the simple counter-based design to encode the label.

The system controller mainly contains the finite state machine (FSM) and the control counter. The design detail of the FSM is shown in Fig. 9. At the beginning, we will pass two parameters into the chip, that is the “mode” and the “training.” The “mode” parameter contains two types, that is, the “conventional mode” and the “O-CIL mode.” If the “training” parameter is given 1, the chip will operate in training mode. Otherwise, it will do only the inferencing subprocess shown in Fig. 9. The control counter mainly controls the number of cycles required in each step shown in Fig. 9. The memory bank stores the parameters required.

**Fig. 9. The finite state machine of the S-QRD-ELM engine. The grey blocks represent the general operations. The orange blocks represent the inference operations. The green blocks represent the learning operations.**
in the S-QRD-ELM, such as the upper triangular matrix $R$, the matrix $u$, and the output weight matrix $\beta$. The CIL module detects whether there is new class data and performs matrix extension for the matrix $u$. The SIMD inference engine executes the matrix-vector multiplication operation of the hidden layer matrix $H$ and the output weight matrix $\beta$ during the inferencing process. We apply the single instruction multiple data (SIMD) architecture with 10 MACs to increase the inferencing throughput. To avoid bottlenecks caused by the SRAM, the $\beta$ SRAM is implemented with high-bandwidth mode which ten weights of $\beta$ are outputted during one read operation. The 1D-DMLA consists of 70 processing elements (PE), which account for the core computation of QRD and BS.

**B. Finite State Machine and Architecture Mapping of Training and Inferencing Process**

The finite state machine of the S-QRD-ELM engine is shown in Fig. 9. It can be separated into three parts: configuration, learning, and inferencing. This FSM orchestrates the modules inside the S-QRD-ELM engine to perform four kinds of modes. Moreover, the modules enabled in each mode are illustrated in Fig. 10.

1) **Online Learning (OL) Mode With and Without CIL:**
First, we have to configure the chip into the learning mode with or without CIL. The $R$ and $u$ matrices inside the memory bank will be initialized with 0. Then the input data and the corresponding labels will be inputted into the chip. The hidden matrix calculator generates the $H$ matrix by calculating the inner product of the input data and the LFSR-generated weights with MUXs. The result is then accumulated with an adder tree and written to the $H$ matrix buffer. If the mode is “O-CIL”, the CIL module performs the existence check of the new class data, remaps the user ID dictionary, extends the $u$ matrix, and finally generates the corresponding one-hot encoded vector $T$. If the mode is “conventional”, the label is processed by the target matrix generator to generate the one-hot encoded vector $T$. The generated $H$ matrix and the one-hot encoded vector $T$ are then input to the 1D-DMLA to perform the QRD operation to generate the $R$ and $u$ matrices and write back the generated $R$ and $u$ matrices to the $R$ and $u$ memories in the memory bank.

2) **Inferencing Mode With and Without CIL:**
Given an input data, the hidden matrix calculator outputs the hidden layer matrix $H$ and stores it in the $H$ matrix buffer. Then the system controller checks whether $R$ and $u$ in the memory bank are updated. If not, the hidden layer matrix $H$ inside the $H$ matrix buffer and the $\beta$ matrix from the memory bank are inputted to the SIMD inference engine to compute the argmax of the output node. If the $R$ and $u$ are updated, the 1D-DMLA first calculates the $\beta$ matrix with the $R$ and $u$ matrices stored in the memory bank and stores it back to the $\beta$ memory. The hidden layer matrix $H$ and the updated $\beta$ matrix from the memory bank are inputted to the SIMD inference engine to calculate the argmax of the output node. If the mode is inferencing with CIL, the internal ID of the predicted node with the max value is inputted into the CIL module and translated to the final predicted label. Otherwise, the predicted internal ID is directly outputted as the final predicted label.

**V. OPTIMIZATIONS INSIDE THE S-QRD-ELM ENGINE**

To further reduce the area and increase the energy efficiency of the S-QRD-ELM engine, we perform several optimizations on the S-QRD-ELM architecture.

**A. From 2D Systolic Array to 1D-DMLA for QRD and BS**

Although the computational complexity inside the S-QRD-ELM is still $O(n^3)$, we can reduce its time complexity to $O(n)$ by using the systolic array (SA) architecture. The SA to solve the S-QRD-ELM is demonstrated in Fig. 11. We map both the QRD and BS computations onto separate SAs to achieve $O(n)$ time complexity. The entire SA can be separated into several parts, including the delayed input, circular boundary cell (BC), squared internal cell (IC), and delayed output. Both the QRD and BS SA will be introduced as follows.
In the QRD SA, it can be separated into two parts, where the matrix $R$ is stored in the left part and the matrix $u$ is stored in the right part. The values in the SA are first initialized. The matrices $H$ and $T$ are reshaped diagonally and inputted into corresponding cells of the SA. The BC projects the vector $(r, x)$ onto the horizontal axis to create $0$, which is required in the triangularization process of QRD. We can obtain the vector $(r', x')$ and the angle $\theta$ evaluating between the vector $(r, x)$ and the horizontal axis from the BC. The angle $\theta$ will be passed horizontally to the next IC to rotate the remaining elements of the vector. The IC in the SA of the QRD phase will rotate the vector $(r', x')$ clockwise with the received angle $\theta$ to the vector $(r'', x'')$. Both the angle $\theta''$ and the output value $x''$ will be passed down to the neighboring ICs. After the entire matrices $H$ and $T$ are inputted, the remaining values inside the SA are the matrices $R$ and $u$.

In the BS SA, the elements of the matrix $R$ are first transposed and stored in the corresponding cells of the SA. After then, the elements of the matrix $u$ are aligned diagonally and inputted into the SA. After $M$ cycles, where $M$ is the number of hidden nodes, we can obtain the elements of the matrix $\beta$ from the right of the SA sequentially. The BC in the BS phase executes the division operation to calculate the unknown variable in the matrix $\beta$, and the solved variable will be passed down to the neighboring IC. The IC substitute the variable received in the neighboring cells and minus it from the corresponding $u$ matrix.

However, directly implement the SA into the engine causes large area and power. The size of the SA depends on the size of the $H$ matrix and the $u$ matrix, which corresponds to the hidden layer number and the output class number. In our specification, where the number of hidden and output nodes are set to 128 and 10, the total number of BCs and ICs is 9536 in the QRD SA and 8256 in the BS SA. It causes a significant area overhead and is not suitable to be implemented on edge devices.

To reduce the area overhead, we condense the SA with proper scheduling and folding techniques as shown in Fig. 12(a). We propose to fold the SA diagonally to reduce the number of total cells. After folding, the adjacent cells are communicated with a 19-bit data path and 1-bit valid and ready control signals. The communication channel between the PEs is controlled by the system controller to achieve proper scheduling. As shown in Fig. 12(b), we take $R \in \mathbb{R}^{3 \times 3}$ and $u \in \mathbb{R}^{3 \times 1}$ as an example to illustrate the proposed scheduling method. The original number of required cells is 9, and the required number of cells is 3 after folding, including 1 BC and 2 ICs. In every clock cycle, the activated cells are marked as blue to complete the computations required in the original SA.

After folding diagonally, our proposed 1D-DMLA only requires 70 cells, including 1 BC and 69 ICs, in both QRD and BS processes, which reduces 99.27% PE usage compared with the original directly mapped SA design. However, such design comes with the cost of half utilization rate (50%) due to the original systolic array is not rectangular.

**B. From Non-Integrated PE Design to Integrated PE Design With Unified-CORDIC**

After the folding techniques above, the number of total cells becomes the same between QRD and BS. Therefore, we utilize the hardware-sharing techniques to design a unified-CORDIC (u-CORDIC) that fits into all the computations required in both QRD and BS.
The cells in the SA can be separated into four kinds: IC for QRD, BC for QRD, IC for BS, and BC for BS. In the QRD process, the BC projects the input vector onto the horizontal axis and the IC rotates the input vector with the given angle. Such computations can be implemented with the CORDIC [17] algorithm and its corresponding architecture. We further modified the CORDIC algorithm to support the linear operations required in both BC and IC of the BS process and proposed the unified-CORDIC algorithm and its architecture design. As operations in u-CORDIC are only composed of addition and shifting operations with iterative executions of these operations, u-CORDIC is suitable to be implemented on hardware.

The u-CORDIC algorithm is divided into two modes and two coordinated operations. Two modes are the angle accumulation mode and the vector rotation mode, respectively. In the angle accumulation mode, the CORDIC projects the vector \((x, y)\) to the horizontal axis and outputs the angle between the vector and the horizontal axis. In the vector rotation mode, the CORDIC is given an angle \(z\), and then rotate the vector \((x, y)\) with the angle \(z\). The operations required in the BC and IC are mapped to the angle accumulation model and vector rotation mode, respectively. The two coordinated operations are the circular-coordinated and the linear-coordinated operations, which can be controlled by the \(a_m\). In addition, in the QRD process, the \(x\) and \(y\) after the vector rotation mode and the angle accumulation mode in the circular-coordinated operation need to be scaled by the scaling factor \(K_m(n)\). The scaling factor is defined as the following formula

\[
K_m(n) = \prod_{i=0}^{n-1} \sqrt{1 + m \mu_i^2 2^{-2i(m,i)}}. \tag{17}
\]

Since the scaling factor can be pre-calculated offline, \(\frac{1}{K_m(n)}\) can be considered as constants. To implement the scaling operations on the same set of hardware, we utilize the canonic signed digit (CSD) code to represent the \(\frac{1}{K_m(n)}\). The CSD code of \(\frac{1}{K_m(n)}\) can be expressed as follows:

\[
\frac{1}{K_m(n)} = \sum_{p=1}^{P} k_p 2^{-i_p}, \tag{18}
\]

where \(k_p = \pm 1\) and \(i_p\) is the index of the non-zero value of the CSD code; hence, we can utilize the shifter and the adder in the CORDIC to achieve the scaling operation.

The integrated PE design with the u-CORDIC engine is shown in Fig. 13. Three additional components are added to enable the linear-coordinated operation of the original CORDIC design. The u-CORDIC is switched between QRD mode and BS mode and between BC mode and IC mode by two additional control signals, the QRD and ANG control signals. Therefore, with the same u-CORDIC, we can operate it into four kinds of PE required in the QRD and BS operation. Compared with the non-integrated PE design, the proposed integrated PE design reduces the area and power by 15.3% and 22.4%, respectively.

### C. CIL Module With Low Area and Power Overhead

As shown in the Fig. 14(a), we first demonstrate implementing the O-CIL on the SA. The mapping of the label to the output node of the ELM and the extension of the \(u\) matrix is the key. Moreover, as illustrated in the previous sections, the main differences between the O-CIL and the OL mode are the detection of the new class data, the mapping between the label and the corresponding output node, and the matrix extension for the node of the new class.

Fig. 14(b) demonstrates the CIL module design. The user ID dictionary contains the mapping table from the external ID to the internal ID, which maps the label to the corresponding output node. The \(u\) memory stores the matrix \(u\), which
contains the information of each class. The CIL controller is responsible for detecting the incoming data of the new class by comparing the given label with the external ID stored in the user ID dictionary. If new class data are detected, the CIL controller extends the output node for the new class by updating the user ID dictionary and duplicating the last row of the matrix with the matrix buffer.

During the learning process of the O-CIL mode, the CIL controller checks whether the external ID exists in the ID dictionary when input data enters. If not, the ID dictionary is remapped and the new external ID is registered. Afterward, the CIL controller extends the matrix stored in the matrix buffer. Finally, the 1D-DMLA updates the matrix and the matrix buffer. During the inference process, after the SIMD inference engine predicts the internal ID, the CIL module transforms the internal ID into the external ID according to the ID dictionary and outputs the final predicted label. As the 1D-DMLA, the buffer and the memory are reused, the overhead of this CIL module is only 4.7% area and 4.95% power of the entire chip.

D. Fixed-Point Analysis

To determine the bit-width inside the S-QRD-ELM and the iteration number required in the u-CORDIC engine. We apply the fixed-point and iteration analysis with the experimental settings of OL in Section III.C to observe the accuracy loss of different bit widths and iterations.

First, we present the fixed-point analysis of the S-QRD-ELM engine. As the implementation of the S-QRD-ELM engine contains the learning process, the bit width inside the engine is critical to its accuracy. After observing the dynamic range among the training procedure on the training set, we set the sign bit as one bit and the integer bit as 9 bits of the S-QRD-ELM engine. Next, we observe the accuracy degradation by setting different fractional bits. As shown in Fig. 15, when the number of fractional bits is more than 9, the accuracy is almost the same as using 32 bits floating point, which is the ideal case. Hence, the final bit-width of the S-QRD-ELM engine is set to 19 bits, including 1 sign bit, 9 integer bits, and 9 fractional bits. Moreover, the bit-width of the matrix used only in the inferencing process can be further reduced to 10 bits to shrink the size of the SIMD inferencing engine and the memory.

Secondly, as the u-CORDIC is the iteration-based method, the number of its iterations matters the precision and thus impacts the final accuracy. Therefore, we present the analysis of the number of CORDIC iterations. As shown in Fig. 16, we observe that the number of iterations in the CORDIC iterative function is at least 10 times to maintain the accuracy of ECG-based user identification. After the iteration analysis, we can calculate the CSD code required for the scaling function. As mentioned in the previous paragraph, the number of fractional bits is 9 and the number of iterations in the u-CORDIC is 10. Therefore, we can rewrite the CSD code of the scaling factor as the following formula

\[
\frac{1}{K_1(10)} = 2^{-1} + 2^{-3} - 2^{-6} - 2^{-9}. \tag{19}
\]

Hence, the CORDIC scaling operation can be achieved by iterating the u-CORDIC four times.

VI. CHIP IMPLEMENTATION AND MEASUREMENT RESULTS

A. Chip Implementation

The S-QRD-ELM engine is fabricated with the TSMC 40-nm 1P9M CMOS process using a standard cell-based design flow. In the process of logic synthesis, since the leakage power of on-chip memory dominates power consumption, the overall chip is synthesized with high threshold voltage (HVT) standard cells. Then, a scan chain is inserted, and the chip can achieve 96.40% of fault coverage. The on-chip memories, such as the memory, are implemented by several macros of single-port static random-access memory (SRAM). During the process of place-and-route, the macros of memory are first placed with soft blockage; then, other standard cells are placed and routed. The chip has 100 pins, including 45 power pads. The I/O domain has a constant standard cells. Then, a scan chain is inserted, and the chip can achieve 96.40% of fault coverage. The on-chip memories, such as the memory, are implemented by several macros of single-port static random-access memory (SRAM). During the process of place-and-route, the macros of memory are first placed with soft blockage; then, other standard cells are placed and routed. The chip has 100 pins, including 45 power pads. The I/O domain has a constant supply voltage of 3.3V. The logic and memory domain is operated at a nominal supply voltage of 0.9V. The chip occupies a 1.33 × 1.33mm² die area and 0.89 × 0.89mm² core area. The chip micrograph and specification of the chip are shown in Fig. 17.

B. Measurement Results

The chip testing environment is illustrated as follows. We use a CQFP-100 test board for chip measurements, and
ELM can be separated into inferencing and training, we select the input layer of the ELM for general machine learning and physical unclonable functions. The final output of the ELM is calculated with digital MACs. Although achieving high energy efficient inferencing process, their chip is not reconfigurable nor supporting OL and O-CIL. The author of [8] proposed an arbitrarily reconfigurable extreme learning machine inference engine with chip implementation using TSMC 40nm. Their ELM engine is separated into two parts, the eigenspace denoising and the ELM inferencing process. In their implementation, they also use LFSRs to generate the input weight and bias matrix. They parallelized the computation of the output matrix with 10 PEs. However, their chip can only execute the inferencing process of ELM, lacking both OL and O-CIL capabilities. Moreover, the robustness comes with the cost of worse energy efficiency.

As for learning of ELM, the authors of [9] proposed a system-on-chip platform based on the OS-ELM algorithm. They designed two sets of hardware to support the initialization and online learning phase of OS-ELM. In the initialization, they applied the Modified Gram-Schmidt QR decomposition (MGS-QRD) to get the matrix $Q$ and finished the calculation of matrix $\beta$ on the CPU of ZYNQ-7000.

During the online learning phase, they simplified Eq. (6) to one data per batch, to eliminate the matrix inversion operation. However, their design requires two sets of hardware which brings overhead in both power and area. Moreover, communications between the processing system and programmable logic are required, which worsens the latency and energy efficiency. Anomaly detection integrated circuit (ADIC) [19] utilized 65-nm CMOS technology to implement the ELM classifier with the OPIUM [20] algorithm. They support up to 7 ELMs with the ensemble techniques to achieve acceptable accuracy. ADIC engine contains 7 base learners (BLs), and each learner includes an input layer module and an output layer module. The input layer module and output layer module contain an adder and a multiplier. They folded the operation required in the eigenspace denoising and the ELM inferencing process. The ADIC engine is separated into two parts, the eigenspace denoising and the ELM inferencing process.
efficiency. Compared with [9], our chip achieves 28.5× learning energy efficiency because our design does not need communication between the processing system and programmable logic. Moreover, only one set of hardware is required during the entire learning process.

VII. CONCLUSION

This paper presents an S-QRD-ELM engine supporting O-CIL for ECG-based user identification. Compared with the BP-NN, this S-QRD-ELM engine outperforms it by 79.7% accuracy in the O-CIL scenario for ECG-based user identification. Compared with the state-of-the-art inferencing-only ELM chip [8], the S-QRD-ELM engine outperforms it by 6.4× inference energy efficiency. Compared with the state-of-the-art FPGA-based OS-ELM accelerator [9], S-QRD-ELM outperforms it by 28.5× learning energy efficiency.

APPENDIX

A. BP-NN-Based Algorithms for Continual Learning and Class Incremental Learning

Continual learning, also known as lifelong learning, never-ending learning, and incremental learning (IL), aims to continuously learn a model with data from new tasks sequentially inputted while preserving knowledge learned from previously learned tasks [21]. When applying deep neural networks (DNNs) trained with iterative backpropagation algorithms in the continual learning scenario, CF easily happens, which makes it difficult to preserve the information learned from the previous tasks. As mentioned in [5], the key cause of CF is the recency bias towards new classes in the last fully connected layer, which is usually the classifier of the model.

Continual learning can be further categorized into three scenarios: task-IL, domain-IL, and class-IL. In task-IL, the task ID of each data is provided during training and inferencing. Therefore, the model only needs to predict the label of the data for a particular task. In domain-IL, the output classes of each task are shared. The model needs to learn different output class distributions of each task without the task ID given. Therefore, domain-IL is more difficult than task-IL. In Class-IL, output classes in each task are different, and the classes between each task are non-overlapping. The task ID is not given in both training and inferencing. Therefore, the model needs to classify which task and output class the data belongs to during inferencing. Therefore, class-IL is the most difficult scenario among continual learning and meets the requirement when learning new classes in the real-world scenario.

Several methods are proposed to train a good DNN model in the restriction of continual learning. These methods can be categorized into regularization, knowledge distillation, and using exemplars. In methods using regularization, Elastic Weight Consolidation (EWC) [22] proposed to modify the loss function when training the DNN as follows:

\[ L(\theta) = L_B(\theta) + \sum_i \frac{\lambda}{2} (\theta_i - \theta^*_A, i)^2, \]

where \( L_B(\theta) \) is the loss function of the new class B, \( \lambda \) is the regularization term of previous classes, \( F \) is the Fisher information matrix, and \( \theta_i \) represents the model’s parameters. EWC protects the accuracy in previous tasks by constraining the parameters with the quadratic penalty and forcing the parameters to have low errors for previous tasks. In methods using knowledge distillation (KD), Learning without Forgetting (LwF) [23] proposed to modify the loss function during the training phase as follows:

\[ L = \lambda L_{old} \left( Y_o, \hat{Y}_o \right) + L_{new} \left( Y_o, \hat{Y}_n \right) + R(\theta), \]

where \( \lambda \) is the loss balance weight, \( L_{old} \left( Y_o, \hat{Y}_o \right) \) is the KD loss of previous classes, \( L_{new} \left( Y_o, \hat{Y}_n \right) \) is the cross-entropy loss of the new classes, and \( R(\theta) \) is the regularization term of the entire model. In methods using exemplars, Incremental Classifier and Representation Learning (iCaRL) [24] proposed to store a small amount of data, that is, the exemplars, from each previous class and update the exemplars with the herding-based selection method. A nearest-mean-of-exemplars classifier is used to get the classification result of the test data.

However, the authors of [4] showed that EWC and LwF could maintain good accuracy only in task-IL and domain-IL. They faced accuracy degradation in the CIL scenario. The iCaRL can maintain good accuracy in the above three scenarios. However, it requires additional memory overhead to store the 2000 exemplars, which does not fit into the online learning scenario.

B. Inversion-Free Algorithms for Learning Process of ELM

The matrix inversion operation in Eq. 6 causes heavy computation overhead and is thus unsuitable for directly implementing the OS-ELM algorithm on hardware. Therefore, inversion-free algorithms are required. Matrix decomposition methods, such as transforming the \( H \) matrix into an orthogonal matrix \( Q \) and an upper triangular matrix \( R \) are introduced. Also, several methods exist for computing the QR decomposition, which can be categorized into Gram–Schmidt, Householder transformation, and Givens rotation. Modified Gram-Schmidt QR decomposition (MGS-QRD) is applied to solve the matrix inversion of the ELM [9], [25]. The inversion matrix of \( H \) can then be calculated with \( R^{-1}Q^T \).

Another inversion-free algorithm is the online pseudoinverse update method (OPIUM) [20]. The core idea of OPIUM is inputting the data-label pair one-by-one instead of batch-by-batch in the original OS-ELM algorithm. Therefore, we can simplify Eq. 6 into Eq. 20, which is the algorithm used in [9]. The above equation does not contain the matrix inversion operation. The OPIUM then rewrites Eq. 5 and Eq. 20 into the following three equations:

\[ \eta_i = \frac{\theta_{i-1}h_i}{1 + h_i^T\theta_{i-1}h_i}, \]

\[ \beta_i = \beta_{i-1} + \eta_i(x_i - \beta_{i-1}h_i), \]

\[ \theta_i = \theta_{i-1} - \eta_i\beta_i h_i. \]

Therefore, we can summarize that OPIUM is the simplified one-by-one solution of the original OS-ELM algorithm. However, this kind of simplification abandoned the parallelism...
operations inside the batch of data. That is, the input data can only be processed one-by-one, which restricts the throughput of the entire system.

REFERENCES


Yi-Ta Chen (Student Member, IEEE) received the B.S. degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 2017, where he is currently pursuing the Ph.D. degree with the Graduate Institute of Electronics Engineering. His research interests include machine learning engine for affective computing, biosignal processing and feature extraction for affective computing, and SW/HW co-design for SDN data plane.

Li-Sheng Chang received the B.S. degree in electronic and computer engineering from the National Taiwan University of Science and Technology, Taipei, Taiwan, in 2019, and the M.S. degree from the Graduate Institute of Electronics Engineering, National Taiwan University, in 2021. His research interests include biomedical signal processing, VLSI architectures, and IC designs.

An-You (Andy) Wu (Fellow, IEEE) received the B.S. degree in electrical engineering from the National Taiwan University (NTU), Taipei, Taiwan, in 1987, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland, College Park, MD, USA, in 1992 and 1995, respectively. In 2000, he joined the Department of Electrical Engineering and the Graduate Institute of Electronics Engineering (GIEE), NTU, as a Faculty Member, where he is currently a Distinguished Professor and has been the Director of GIEE since 2016. From 2007 to 2009, he was on leave from NTU and worked as the Deputy General Director of the SoC Technology Center (STC), Industrial Technology Research Institute (ITRI), Hsinchu, Taiwan. From 2016 to 2019, he worked as the Director of GIEE, NTU. His research interests include VLSI architectures for signal processing and communications and adaptive/multi-rate signal processing. He has published more than 250 refereed journals and conference papers in the above research areas, together with five book chapters and 20 granted U.S. patents. He was elevated to IEEE Fellow for his contributions to DSP algorithms and VLSI designs for communication IC/SoC in 2015.