FPGA Implementation of Orthogonal Matching Pursuit for Compressive Sensing Reconstruction

Article *in* IEEE Transactions on Very Large Scale Integration (VLSI) Systems · October 2015

DOI: 10.1109/TVLSI.2014.2358716

Abbes Amira
University of Sharjah
381 PUBLICATIONS **4,538** CITATIONS
SEE PROFILE

Basant Kumar Mohanty
Narsee Monjee Institute of Management Studies
62 PUBLICATIONS **1,082** CITATIONS
SEE PROFILE

Somaya Al-ma'adeed
Qatar University
238 PUBLICATIONS **3,363** CITATIONS
SEE PROFILE

FPGA Implementation of Orthogonal Matching Pursuit for Compressive Sensing Reconstruction

Hassan Rabah, *Senior Member, IEEE*, Abbes Amira, *Senior Member, IEEE*, Basant Kumar Mohanty, *Senior Member, IEEE*, Somaya Almaadeed, *Senior Member, IEEE*, and Pramod Kumar Meher, *Senior Member, IEEE*

***Abstract*— In this paper, we present a novel architecture based on field-programmable gate arrays (FPGAs) for the reconstruction of compressively sensed signal using the orthogonal matching pursuit (OMP) algorithm. We have analyzed the computational complexities and data dependence between different stages of OMP algorithm to design its architecture that provides higher throughput with less area consumption. Since the solution of least square problem involves a large part of the overall computation time, we have suggested a parallel low-complexity architecture for the solution of the linear system. We have further modeled the proposed design using Simulink and carried out the implementation on FPGA using Xilinx system generator tool. We have presented here a methodology to optimize both area and execution time in Simulink environment. The execution time of the proposed design is reduced by maximizing parallelism by appropriate level of unfolding, while the FPGA resources are reduced by sharing the hardware for matrix–vector multiplication across the data-dependent sections of the algorithm. The hardware implementation on the Virtex6 FPGA provides significantly superior performance in terms of resource utilization measured in the number of occupied slices, and maximum usable frequency compared with the existing implementations. Compared with the existing similar design, the proposed structure involves 328 more DSP48s, but it involves 25 802 less slices and 1.85 times less computation time for signal reconstruction with $N = 1024$, $K = 256$, and $m = 36$, where N is the number of samples, K is the size of the measurement vector, and m is the sparsity. It also provides a higher peak signal-to-noise ratio value of 38.9 dB with a reconstruction time of 0.34 μs, which is twice faster than the existing design. In addition, we have presented a performance metric to implement the OMP algorithm in resource constrained FPGA for the better quality of signal reconstruction.**

***Index Terms*— Compressive sensing, field-programmable gate array (FPGA) implementation, hardware reconstruction, low-complexity architecture, orthogonal matching pursuit (OMP) algorithm, system-level modeling.**

Manuscript received July 30, 2013; revised July 13, 2014; accepted September 4, 2014. Date of publication October 2, 2014; date of current version September 23, 2015.

H. Rabah is with the Institute Jean Lamour, University of Lorraine, Nancy F-54000, France (e-mail: hassan.rabah@univlorraine.fr).

A. Amira is with the Department of Computer Science and Engineering, Qatar University, Doha 2713, Qatar, and also with the School of Engineering and Computer Science, University of the West Scotland, Paisley PA1 2BE, U.K. (e-mail: abbes.amira@uws.ac.uk).

B. K. Mohanty is with the Department of Electronics and Communication Engineering, Jaypee University of Engineering and Technology, Guna 473226, India (e-mail: bk.mohanti@juet.ac.in).

S. Almaadeed is with the Department of Computer Science and Engineering, Qatar University, Doha 2713, Qatar (e-mail: s_alali@qu.edu.qa).

P. K. Meher is with the School of Computer Engineering, Nanyang Technological University, Singapore 639798 (e-mail: aspkmeher@ntu.edu.sg).

Digital Object Identifier 10.1109/TVLSI.2014.2358716

I. Introduction

Compressed sensing (CS) allows a signal to be acquired and accurately reconstructed with significantly fewer samples than those required by the sampling constraint provided by the Nyquist rate [1]. It opens a radically new era in the domain of signal acquisition. CS relies on the sparsity of a signal and/or its optimal representation in an appropriate transform domain. Sparsity of signal is found abundantly in wide and diverse kind of signals and very much prevalent in biosignals, medical images, and radar signals [2], [3]. CS is generally performed by multiplying the input signal by a *measurement matrix*. The reconstruction of original signal from compressively sampled signal requires the knowledge of this matrix in advance. The reconstruction process consists of finding the best solution to an underdetermined system of linear equations given by $\mathbf{y} = \mathbf{\Phi x}$, where the measurement matrix $\mathbf{\Phi}$ and the measured signal $\mathbf{y}$ are known. No knowledge of the original signal $\mathbf{x}$ is required for reconstruction except that it is sparse in the sampled domain.

Various algorithms have been proposed for the reconstruction of signals from the compressively sensed samples. Matching pursuit (MP) [4] is a common approach for sparse signal reconstruction, which greedily computes an approximation to the original signal [5]. MP algorithm iteratively identifies the column of measurement matrix that is most correlated to a current signal estimate, followed by a simple update that computes an improved signal estimate. While each iteration of MP requires very low computational effort, the number of iterations heavily depends on the sparsity level m, and consequently, MP is more suitable for signals with high sparsity degrees [4], [6]. Orthogonal MP (OMP) proposed in [7] is a more complex algorithm that incorporate a least-squares (LS) step to compute a signal estimate. In OMP, the LS step significantly reduces the number of required iterations compared with MP, but it results in a high computational complexity per iteration [6]. This complexity is mainly due to the large number of inner-product computation (IPC), several comparison operations, and matrix inversion. Therefore, the high computational complexity of OMP algorithm is a major concern for its implementation to achieve real-time reconstruction of compressively sensed signals.

Several software implementations on general-purpose computer and graphic processor unit (GPU) have been proposed in the literature. It is observed that the acceleration achieved by the GPU-based implementation is significantly better than CPU-based implementation [8]. However, the GPU-based implementation has a major problem of intermittent memory bandwidth between the main memory and the GPU, which does not facilitate regular flow of data communication with the host [8], [9]. Several schemes have been proposed to accelerate individual computing stages of OMP algorithm in the hardware solutions presented in [10]–[12]. We found only a few proposed designs for the complete implementation of OMP algorithm in hardware. Recently, Septimus and Steinberg [13] and Stanislaus and Mohsenin [14] have presented a field-programmable gate array (FPGA) implementation of OMP algorithm. However, a close examination of the algorithm and the proposed architectures reveals that those are from MP implementation rather than OMP, as stated by the authors. Septimus and Steinberg [13] have proposed an FPGA-based design that involves significantly higher cycle period due to a large-size inner product (IP) in the critical path. Stanislaus and Mohsenin [14] and Mohsenin and Stanislaus [15] have also proposed an FPGA implementation of MP algorithm based on QR-decomposition scheme for matrix inversion to reduce the computation complexity. Blaché *et al.* [16] and Bai *et al.* [17] have presented FPGA implementations of reconstruction algorithms based on OMP. In the design presented in [16], the matrix inversion is based on CORDIC divider with high latency and a sequential execution of several parts of matrix multiplication. Bai *et al.* [17] have presented a hardware design of OMP algorithm based on QR decomposition and consisting of a vector multiplication units with multiple operation modes and a square root reciprocal. The reconstruction time of this design is 0.63 ms for $N = 1024$, $K = 256$, and $m = 36$.

In the existing implementations of both MP and OMP algorithms, the IPC performed during the search of nonredundant columns of $\mathbf{\Phi}$ takes an important number of cycles and is generally optimized by exploiting inherent parallelism. The simultaneous access to data is an important issue for an efficient implementation of a parallel structure, which requires an adequate memory organization. One of the most critical part of MP and OMP algorithms is solving the LS problem (LSP). However, while the LSP is solved once in the final stage of MP algorithm, it should be solved in each iteration of the OMP algorithm. The LSP involves several matrix arithmetics, particularly, the inversion, which requires division operation and significantly affects the cycle period and computational latency. In this paper, we have addressed all these challenges by presenting an efficient pipelined architecture to implement the OMP algorithm. In addition, we have proposed an efficient design for IPC and inversion based on Newton–Raphson iteration, and a scalable pipelined architecture of Moore–Penrose pseudoinverse for low-latency solution of LSP. We have also presented a Simulink-based design flow for the implementation of the proposed OMP architecture. The proposed architecture is also scalable and can be used for unknown sparsities. The rest of this paper is organized as follows. In Section II, we have given a detailed description of the OMP algorithm. The proposed hardware architecture is described in Section III. FPGA implementation flow, the synthesis results, and the performance of the proposed design are presented in Section IV. The conclusion and the scope of the future work are outlined in Section V.

Algorithm 1: OMP Algorithm for Signal Recovery

```
Input   : $\mathbf{\Phi} \in \mathbb{R}^{K\times N}$: The sampling matrix
          $\mathbf{y} \in \mathbb{R}^{K}$: The measurements vector
          $m$: The sparsity level of the signal $\mathbf{x}$
Output  : $\tilde{\mathbf{x}} \in \mathbb{R}^{N}$: The estimate of the original signal
Procedure:
  1) Initialize the residual $\mathbf{r}_0 = \mathbf{y}$, and the index set
     $\Lambda_0 = \{\emptyset\}$. Set iteration counter $i = 1$
  2) Find the index $\lambda_i$ solving the optimization problem
     $\lambda_i = \arg\max_{j=1\ldots N} |\langle r_{i-1}, \phi_j\rangle|$
  3) Augment the index set $\mathbf{\Lambda}_i = \mathbf{\Lambda}_{i-1} \cup \{\lambda_i\}$ and the
     matrix of chosen columns $\tilde{\mathbf{\Phi}}_i = [\tilde{\mathbf{\Phi}}_{i-1} \;\; \phi_{\lambda_i}]$.
  4) Solve a least square problem for new estimate $\tilde{\mathbf{x}}$ of
     signal $\mathbf{x}$:
     $\tilde{\mathbf{x}}_i = \arg\min_{x} \|\mathbf{y} - \tilde{\mathbf{\Phi}}_i\mathbf{x}\|$
  5) Compute the new residual :
     $\mathbf{r}_i = \mathbf{y} - \tilde{\mathbf{\Phi}}_i\tilde{\mathbf{x}}_i$
  6) Increment counter $i$ and return to step-2 if $i < m$.
  7) Retrieve the finale estimate $\tilde{\mathbf{x}}$.
```

II. Overview of OMP Algorithm and Its Complexity Analysis

CS is based on the fact that the information from a signal may be captured by a small set of nonadaptive linear measurements when the signal is sparse in some basis [1]. An m-sparse signal vector consists of at most m nonzero scalar components. A signal vector $\mathbf{x} \in \mathbb{R}^N$ acquired via linear measurements is given by

$$\mathbf{y} = \mathbf{\Phi}\mathbf{x} + \mathbf{n} \qquad (1)$$

where $\mathbf{\Phi} \in \mathbb{R}^{K.N}$ is a rectangular sampling matrix modeling the sampling system, $\mathbf{y} \in \mathbb{R}^K$ is the measurement vector, and $\mathbf{n}$ is a K-point vector that represents the measurement error or noise. The columns of matrix $\mathbf{\Phi}$ denoted $\phi_1, \phi_2, \ldots, \phi_N$ are K-point vectors ($K < N$), also called atoms. The length of measurements vector $\mathbf{y}$ is in general assumed to be much smaller than the length of signal vector $\mathbf{x}$.

A. General Description of OMP Algorithm

The OMP algorithm proposed in [7] is given in Algorithm 1. It takes the measurement matrix $\mathbf{\Phi}$ and the measured vector $\mathbf{y}$ as inputs and provides an estimate $\tilde{\mathbf{x}}$ of the original signal $\mathbf{x}$. This algorithm is iterative. During each iteration, it chooses one of the columns of $\mathbf{\Phi}$, which is most strongly correlated with the residual of measurements $\mathbf{y}$, and then it removes the contribution of this column to compute a new residual. It also computes a new estimate of the original signal; after

m iterations, the algorithm will generate the finale estimate of the original signal.

The optimization problem of step 2 of Algorithm 1 is solved by calculating correlation vector $\mathbf{w}$ as follows:

$$\mathbf{w} = \mathbf{\Phi}^T \mathbf{r}_{i-1} \tag{2}$$

where $\mathbf{r}_{i-1}$ is the residual vector of $(i-1)$th iteration. The index λ_i of the component of $\mathbf{w}$ having maximum absolute value is identified, and the corresponding column is extracted from $\mathbf{\Phi}$ to constitute matrix $\tilde{\mathbf{\Phi}}$ of such extracted columns. According to step 4 of Algorithm 1, an estimate of the reconstructed signal $\tilde{\mathbf{x}}_i$ is obtained by solving the following:

$$\mathbf{y} = \tilde{\mathbf{\Phi}}\tilde{\mathbf{x}} \tag{3}$$

where $\tilde{\mathbf{\Phi}}$ is a $(K \times m)$ rectangular matrix with $K > m$. A common approach to invert such rectangular matrix is to use the Moore–Penrose pseudoinverse, expressed by

$$\tilde{\mathbf{\Phi}}^{\dagger} = \left(\tilde{\mathbf{\Phi}}^T \tilde{\mathbf{\Phi}}\right)^{-1} \tilde{\mathbf{\Phi}}^T \tag{4}$$

where $\tilde{\mathbf{\Phi}}^{\dagger}$ is the pseudoinverse of Moore–Penrose. Therefore, the solution of (3) is obtained by solving the following:

$$\mathbf{w} = \mathbf{C}\tilde{\mathbf{x}} \tag{5}$$

where $\mathbf{C} = \tilde{\mathbf{\Phi}}^T \tilde{\mathbf{\Phi}}$ is a symmetric matrix $\in \mathbb{R}^{m \times m}$. Equation (5) can be solved by matrix inversion or by forward/backward substitution [12].

Various methods can be used to find the inverse of a matrix, such as Cholesky factorization, **LU**, and **QR** decomposition methods. We have used the modified Cholesky factorization method in [18] for matrix inversion since it does not require square-root operations. Based on the modified Cholesky factorization, matrix $\mathbf{C}$ can be expressed as the product of three matrices as

$$\mathbf{C} = \mathbf{L}\mathbf{D}\mathbf{L}^T. \tag{6}$$

The lower triangular matrix $\mathbf{L}$ and diagonal matrix $\mathbf{D}$ are computed using the following relations:

$$L_{i,j} = \frac{1}{D_{j,j}}\left\{C_{i,j} - \sum_{k=1}^{j-1}(L_{i,k}L_{j,k}D_{k,k})\right\}, \quad i > j \tag{7a}$$

$$D_{i,i} = C_{i,i} - \sum_{k=1}^{i-1}\left(L_{i,k}^2 D_{k,k}\right). \tag{7b}$$

The inverse of matrix $\mathbf{C}$ is obtained as follows:

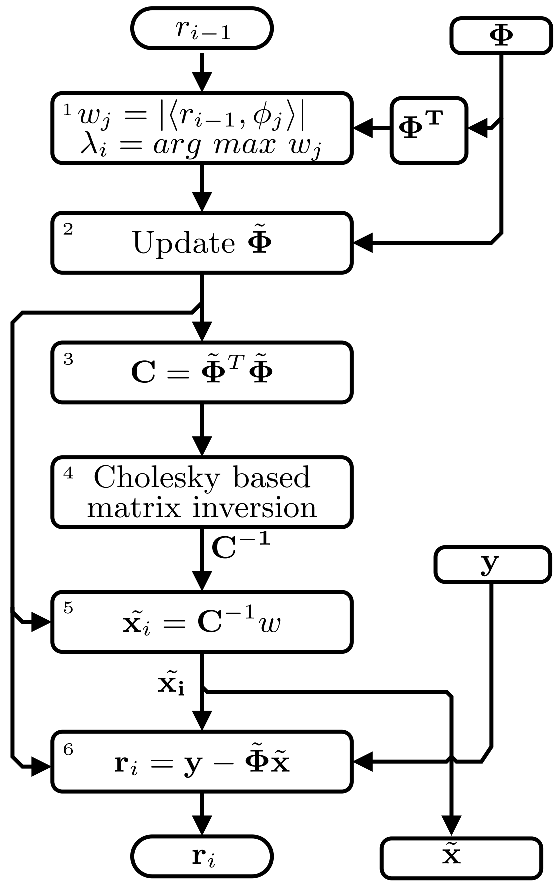
$$\mathbf{C}^{-1} = (\mathbf{L}^{-1})^T \mathbf{D}^{-1} \mathbf{L}^{-1}. \tag{8}$$

The inversion of matrix $\mathbf{D}$ is obtained by taking inversion of its diagonal components, while the inversion of matrix $\mathbf{L}$ is performed iteratively using the relation

$$L_{i,j}^{-1} = -\sum_{k=j}^{i-1} L_{i,k}L_{k,j}^{-1} \tag{9}$$

for $i > j$. In step 6 of Algorithm 1, the residue vector $\mathbf{r}$ is updated for the next iteration using the relation

$$\mathbf{r}_i = \mathbf{y} - \tilde{\mathbf{\Phi}}\tilde{\mathbf{x}}. \tag{10}$$



r_{i-1}
Φ
$^1 w_j = |\langle r_{i-1}, \phi_j \rangle|$
$\lambda_i = arg\ max\ w_j$
Φ^T
^{2} Update $\tilde{\Phi}$
$^3\ C = \tilde{\Phi}^T\tilde{\Phi}$
^{4} Cholesky based matrix inversion
C^{-1}
y
$^5\ \tilde{x}_i = C^{-1}w$
$\tilde{x}_i$
$^6\ r_i = y - \tilde{\Phi}\tilde{x}$
r_i
$\tilde{x}$

Fig. 1. Flow graph of one iteration of OMP algorithm. For MP algorithm, functions 3 and 4 are executed once outside the loop and function 6 consists of computing $\mathbf{r}_i = \mathbf{r_{i-1}} - \tilde{\mathbf{\Phi}}\tilde{\mathbf{\Phi}}^T\mathbf{r_{i-1}}$.

To apply backward/forward substitution approach, it is also necessary to decompose matrix $\mathbf{C}$ into three matrices $\mathbf{L}$, $\mathbf{D}$, and $\mathbf{L^T}$, as shown in (6). Solving (5) amounts to solving three equations: $\mathbf{Lz} = \mathbf{w}$ using forward substitution, $\mathbf{Dr} = \mathbf{z}$, and $\mathbf{L^T x} = \mathbf{r}$ using backward substitution, where $\mathbf{z}$ and $\mathbf{r}$ are intermediate vectors.

Matrix $\mathbf{C}$ is symmetric, and its size grows each iteration from 1×1 to $m \times m$. A simple analysis shows that the calculation of $\mathbf{L}$ and $\mathbf{D}$ takes $m(m+1)/2$ cycles for m iterations and that of $\mathbf{L^{-1}}$ and $\mathbf{D^{-1}}$ can be performed in parallel and takes the same number of cycles. As will be demonstrated later in this paper, the calculation of $\mathbf{C^{-1}}$ will take $(m+1)m/2$ and that of $\mathbf{C^{-1}w}$ can take 1, m, or m^2, depending on the degree of parallelism of the adopted architecture. In our case, with m cycles, we obtain a total number of cycles equal to $(m+2)m$ for matrix inversion approach. In the second approach, the calculation of $\mathbf{L}$ and $\mathbf{D}$, $\mathbf{z}$, and $\mathbf{x}$ will take $(m+1)m/2$ cycles each, which corresponds to a total of $3(m+1)m/2$. For this reason, we have chosen to develop the first approach that has more potential of parallelism. Moreover, we adopted this approach for comparison with similar proposed architectures [13], [17].

B. Complexity Analysis of OMP Algorithm

The OMP algorithm is iterative and consists of six ordered functions. We have presented these functions and data dependencies between them in the data-flow graph shown in Fig. 1.

TABLE I

COMPUTATION COMPLEXITY OF OMP ALGORITHM (m DEGREE OF SPARSITY, K SIZE OF MEASUREMENT VECTOR, AND N NUMBER OF SAMPLES)

Functions \ Operations		Multiplication	Addition/subtraction	Comparison	Negate	Division
Function-1		NKm	$N(K-1)m$	0	0	0
Function-2		0	0	$(N-1)m$	0	0
Function-3		$K(m+1)m/2$	$(K-1)(m+1)m/2$	0	0	0
Function-4	**L**	$(m-1)m/2$	0	0	0	0
	D	$(m^3-m)/6$	$(m^3-m)/6$	0	0	0
	$\mathbf{D}^{-1}$	0	0	0	0	m
	$\mathbf{L}^{-1}$	$(m^3-3m^2+2m)/6$	$(m^3-3m^2+2m)/6$	0	$m(m-1)/2$	0
	$\mathbf{C}^{-1}$	$(2m^3+3m^2-5m)/6$	$(m^3-m)/6$	0	0	0
	Subtotal	$(4m^3+3m^2-7m)/6$	$(m^3-m^2)/2$		$m(m-1)/2$	m
Function-5		$(2m^3+3m^2-5m)/6$	$(m^3-m)/6$	0	0	0
Function-6		$Km(m+1)/2$	$K(m+1)m/2$	0	0	0

The computation of the six functions is performed sequentially and repeated for each iteration. The functions 1 and 2 are the computation of the correlation vector (2) followed by the comparison operation to extract the column of $\mathbf{\Phi}$ having maximum correlation and then update the matrix $\tilde{\mathbf{\Phi}}$ by augmenting such extracted columns to construct the measurement-generation matrix $\tilde{\mathbf{\Phi}}$. In functions 3 and 4, matrix $\mathbf{C}$ is reconstructed from matrix $\tilde{\mathbf{\Phi}}$, and then its inverse is computed using Cholesky decomposition method given in (7)–(9). Finally, an estimate $\tilde{\mathbf{x}}$ of the reconstructed signal $\mathbf{x}$ is computed according to (5), and then the residual vector is updated for the next iteration.

We have estimated the complexity of these functions involved in the OMP algorithm (Table I). The total number of multiplications, additions, comparisons, negates, and divisions is, respectively, $7m^3/6 + (K+1)m^2 + (NK+K-2)m$, $4m^3/6 + (K-1)m^2 + (NK-N-2/3)m$, $(N-1)m$, $m(m-1)/2$, and m. As $N > K \gg m$, we can observe from Table I that functions 1, 3, and 6 present the most important part of the overall complexity of OMP algorithm.

The main problem associated in accelerating the OMP algorithm is that due to its iterative nature, it does not allow parallel execution. In addition, the computational flow of the six functions in each iteration of the algorithm is data dependent. Therefore, the computation of six functions cannot be performed concurrently. However, the processing can be speeded up by introducing parallelism in some functions selectively. We can perform all the N IPC in parallel and perform fast comparison of the IP values to identify the maximally correlated column of $\mathbf{\Phi}$. This will require huge resources. Therefore, we have designed a pipeline IPC unit that performs one IPC followed by a comparison in each cycle. The functions 1, 3, and 6 utilize the same IP unit, thus leading to a significant resource optimization. We find from the existing designs that Cholesky decomposition requires significantly large number of clock cycles while it has less computation than other functions of OMP algorithm. This is mainly due to the division operation and the complex data dependency of the components of matrices $\mathbf{L}$ and $\mathbf{D}$. Hence, in this paper, we present an optimized parallel structure for matrix inversion based on Cholesky decomposition. The description of these structures is presented in the following section along with the detailed design of the proposed architecture for OMP algorithm.

III. PROPOSED ARCHITECTURE

The proposed structure for the implementation of OMP algorithm is shown in Fig. 2. It has four computing blocks: 1) K-point inner product and comparator unit (K-IPCU); 2) Cholesky inversion unit (CIU); 3) residual computation unit; and 4) reconstructed signal computation unit. In addition, it has memory and control units. The measurement matrix $\mathbf{\Phi}$, measurement vector $\mathbf{y}$, and all intermediate matrices and vectors are stored in the memory units. In the K-IPCU, the K-point IP, which is shared between operations 1, 3, and 6, receives columns of measurement matrix $\mathbf{\Phi}$ and residual vector $\mathbf{r}_{i-1}$ and computes matrix $\mathbf{C}$ or correlation vector $\mathbf{W}$, or $\tilde{\mathbf{\Phi}}\tilde{\mathbf{x}}$. The K-IPCU also performs the necessary comparison to identify the column of $\mathbf{\Phi}$ that is maximally correlated with the residual vector of the previous iteration. The results of these computations are stored in memory units. Similarly, the CIU receives the components of intermediate matrix $\mathbf{C}$ from the memory units to calculate its inverse. The control unit (not shown in Fig. 2) generates the address values to perform read and write operations on the memory unit, and sends the necessary control signals to the computing units to initiate their operations.

A. Proposed Structure of the K-IPCU

The structure of K-IPCU is shown in Fig. 3 for $N = 1024$, $K = 256$, and $m = 36$. It is composed of four 64-point inner products (64-point IPs), a comparator, a dispatcher, and an adder tree. The four 64-point IP is identical and composed of 64 parallel multipliers and 63 adders. The dispatcher is used to transfer the adequate data to the four 64-point IP. Before the execution starts, the residual vector $\mathbf{r}_{i-1}$ is initialized by the input data vector $\mathbf{y} \in \mathbb{R}^K$. The vector $\mathbf{r}_{i-1}$ is stored in a RAM so that any of the K values can be accessed in one clock cycle. The matrix $\mathbf{\Phi} \in \mathbb{R}^{K\times N}$ is also stored in a RAM

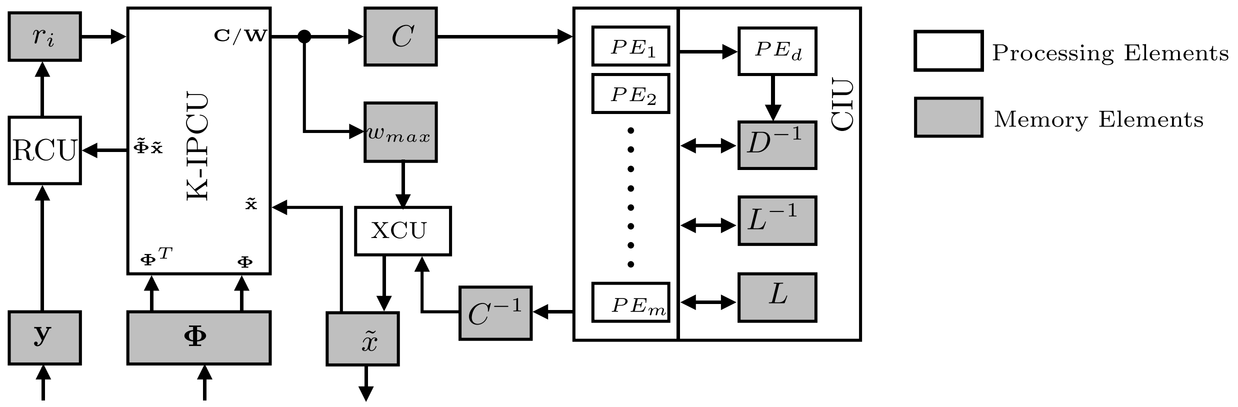


Fig. 2. Proposed structure for OMP implementation.

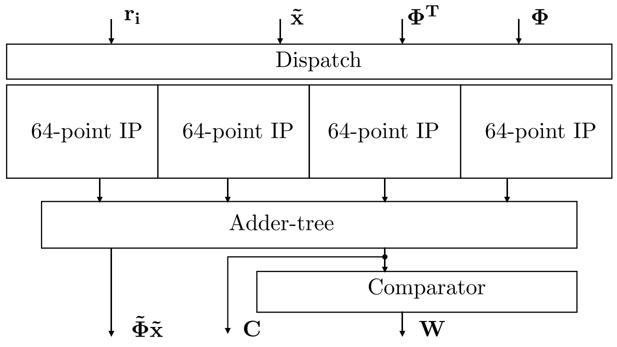


Fig. 3. Structure of K-IPCU for $N = 1024$, $K = 256$, and $m = 36$.

of K modules allowing the access to a complete vector $\phi_n \in \mathbb{R}^K$ ($n = 1, \ldots, N$) in one clock cycle. During the first N cycles of each iteration for the computation of K-IPCU, N columns of $\mathbf{\Phi}$ are retrieved from the memory unit in serial order and fed to the K-IPCU. During the ith iteration, the K-IPCU performs the IP of residual vector r_{i-1} with a column of $\mathbf{\Phi}$ available from the memory in each cycle of N cycles. During nth clock cycle of the first N cycles of ith iteration, the K-IPCU performs IP of ϕ_n with $\mathbf{r}_{i-1}$ and the product value is sent to the comparator to find the maximum absolute of IP among such successive N values.

The comparison operation is concurrently performed with the IP. The comparator sends the column index λ_i of $\mathbf{\Phi}$ that results in the highest correlation, which in turn writes the specified column of $\mathbf{\Phi}$ to the measurement-generator matrix $\tilde{\mathbf{\Phi}}$. After the search operation on the matrix $\mathbf{\Phi}$ during N cycles of every iteration, the construction of the matrix $\mathbf{C}$ is performed by matrix–matrix multiplication between the transpose matrix $\tilde{\mathbf{\Phi}}^T$ and $\tilde{\mathbf{\Phi}}$. We have used the same K-IPCU to perform this matrix multiplication. For this, the K-IPCU receives one row of the transpose matrix $\tilde{\mathbf{\Phi}}^T$ and one column of $\tilde{\mathbf{\Phi}}$ from the memory unit and computes one component of matrix in one cycle. As the matrix $\mathbf{C}$ is a symmetric matrix, it is not necessary to compute all its components. The size of matrix $\mathbf{C}$ increases from 1×1 in the first iteration to $m \times m$ in the mth iteration. Therefore, the component of $\mathbf{C}$ is obtained in $m(m+1)/2$ cycles for m iterations. Matrix $\mathbf{C}$ is stored in the memory unit to be used for calculation of its inverse.

B. Proposed Structure for CIU

Since matrix $\mathbf{C}$ is symmetric, only the diagonal and lower triangular values are required by the CIU to calculate the inverse of $\mathbf{C}$. Due to the division operation and complex inter data dependency of lower triangular matrix $\mathbf{L}$ and diagonal matrix $\mathbf{D}$, a straightforward implementation of Cholesky decomposition involves relatively more number of clock cycles than other computing units of the OMP algorithm. Based on our dependence analysis, we have proposed a maximally parallel structure for CIU and pipelined the structure wherever is necessary.

1) Matrix Decomposition: In Cholesky decomposition, the square matrix $\mathbf{C}$ is decomposed into lower triangular matrix $\mathbf{L}$ and diagonal matrix $\mathbf{D}$. The elements of $\mathbf{L}$ and $\mathbf{D}$ are calculated using (7). It can be observed from (7) that matrices $\mathbf{L}$ and $\mathbf{D}$ are interdependent. Therefore, the components of $\mathbf{L}$ and $\mathbf{D}$ are calculated in a specific order.

The elements of matrix $\mathbf{C}$ as well as those of $\mathbf{L}$ and $\mathbf{D}$ are computed iteratively. A full parallel implementation is not necessary in this case. To demonstrate this, we have derived separate relations for individual components of each matrix $\mathbf{L}$ and $\mathbf{D}$ to map the computation in each iteration for $m = 5$ as follows.

Iteration 1

$$D_{11}^{-1} = C_{11}^{-1}. \tag{11a}$$

Iteration 2

$$D_{21} = C_{21} \tag{12a}$$

$$L_{21} = D_{11}^{-1} \cdot D_{21} \tag{12b}$$

$$D_{22} = (C_{22} - L_{21} \cdot D_{21}). \tag{12c}$$

Iteration 3

$$D_{31} = C_{31} \tag{13a}$$

$$L_{31} = D_{11}^{-1} \cdot D_{31} \tag{13b}$$

$$D_{32} = C_{32} - L_{21} \cdot D_{31} \tag{13c}$$

$$L_{32} = D_{22}^{-1} \cdot D_{32} \tag{13d}$$

$$D_{33} = (C_{33} - L_{31} \cdot D_{31}) - L_{32} \cdot D_{32}. \tag{13e}$$

	Iterations	1	2			3				4					5					
C_{i1}	PE_d	D_{11}^{-1}			D_{22}^{-1}				D_{33}^{-1}					D_{44}^{-1}						D_{55}^{-1}
C_{i2}	$PE1$		L_{21}	D_{22}		L_{31} D_{32}				L_{41} D_{42}					L_{51} D_{52}					
C_{i3}	$PE2$						L_{32} d_{33}^{1}	D_{33}		d_{43}^{1}	L_{42} D_{43}				d_{53}^{1}	L_{52} D_{53}				
C_{i4}	$PE3$										d_{44}^{1}	L_{43} d_{44}^{2}	D_{44}		d_{54}^{1}	d_{54}^{2}	L_{53} D_{54}			
C_{i5}	$PE4$															d_{55}^{1}	d_{55}^{2}	L_{54} d_{55}^{3}	D_{55}	

Fig. 4. Data dependency and scheduled tasks between PEs. D_{ij} and d_{ij}^{k} are temporal variables computed by a PE and used locally or passed to the adjacent PE.

Iteration 4

$$D_{41} = C_{41} \tag{14a}$$
$$L_{41} = D_{11}^{-1} \cdot D_{41} \tag{14b}$$
$$D_{42} = C_{42} - L_{21} \cdot D_{41} \tag{14c}$$
$$L_{42} = D_{22}^{-1} \cdot D_{42} \tag{14d}$$
$$D_{43} = (C_{43} - L_{31} \cdot D_{41}) - L_{32} \cdot D_{42} \tag{14e}$$
$$L_{43} = D_{33}^{-1} \cdot D_{43} \tag{14f}$$
$$D_{44} = ((C_{44} - L_{41} \cdot D_{41}) - L_{42} \cdot D_{42}) - L_{43} \cdot D_{43}. \tag{14g}$$

Iteration 5

$$D_{51} = C_{51} \tag{15a}$$
$$L_{51} = D_{11}^{-1} \cdot D_{51} \tag{15b}$$
$$D_{52} = C_{52} - L_{21} \cdot D_{51} \tag{15c}$$
$$L_{52} = D_{22}^{-1} \cdot D_{52} \tag{15d}$$
$$D_{53} = (C_{53} - L_{31} \cdot D_{51}) - L_{32} \cdot D_{52} \tag{15e}$$
$$L_{53} = D_{33}^{-1} \cdot D_{53} \tag{15f}$$
$$D_{54} = ((C_{54} - L_{41} \cdot D_{51}) - L_{42} \cdot D_{52}) - L_{43} \cdot D_{53} \tag{15g}$$
$$L_{54} = D_{44}^{-1} \cdot D_{54} \tag{15h}$$
$$D_{55} = (((C_{55} - L_{51} \cdot D_{51}) - L_{52} \cdot D_{52}) - L_{53} \cdot D_{53}) - L_{54} \cdot D_{54}. \tag{15i}$$

From this decomposition, we can note that for all the iterations $i > 1$, the couples (L_{i1}, D_{i2}) are computed using similar equation involving multiplication and subtraction. This observation is valid for the couples (L_{i2}, D_{i3}), (L_{i3}, D_{i4}), and (L_{i4}, D_{i5}). D_{ij} are computed in iteration i by PEj and passed to the adjacent processing element (PE) (if exists). In FPGA circuits, the multipliers and subtractors can be implemented using the DSP48 blocks [19]. Fig. 5 shows the proposed structure of the PE exploiting the DSP48 resources for efficient computation of the elements of **L** and **D**.

2) Inversion of Matrix **L***:* The inversion of matrix **L** is performed iteratively, as shown in (16) for $m = 5$. In each iteration i, only the elements L_{ij}^{-1} are computed with $j \in [1, i-1]$ and $i > 1$

$$L_{21}^{-1} = -L_{21} \tag{16a}$$
$$L_{31}^{-1} = -(L_{31} + L_{32} \cdot L_{21}^{-1}) \tag{16b}$$
$$L_{32}^{-1} = -L_{32} \tag{16c}$$
$$L_{41}^{-1} = -(L_{41} + L_{42} \cdot L_{21}^{-1} + L_{43} \cdot L_{31}^{-1}) \tag{16d}$$
$$L_{42}^{-1} = -(L_{42} + L_{43} \cdot L_{32}^{-1}) \tag{16e}$$

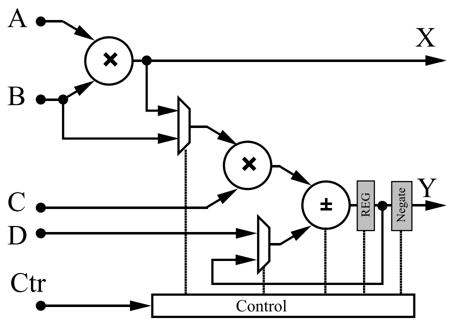


Fig. 5. Architecture of PEs (PE1, ..., PEm).

$$L_{43}^{-1} = -L_{43} \tag{16f}$$
$$L_{51}^{-1} = -(L_{51} + L_{52} \cdot L_{21}^{-1} + L_{53} \cdot L_{31}^{-1} + L_{54} \cdot L_{41}^{-1}) \tag{16g}$$
$$L_{52}^{-1} = -(L_{52} + L_{53} \cdot L_{32}^{-1} + L_{54} \cdot L_{42}^{-1}) \tag{16h}$$
$$L_{53}^{-1} = -(L_{53} + L_{54} \cdot L_{43}^{-1}) \tag{16i}$$
$$L_{54}^{-1} = -L_{54}. \tag{16j}$$

As shown in (16), for each iteration, the elements L_{ij}^{-1} are computed using the results of the previous iterations. The operations involved are addition, multiplication, and arithmetic negation. The proposed PE shown in Fig. 5 is configured to compute the inverse of matrix **L** using negate, multiply, and add operators. The inverse of matrix **L** is computed in parallel with the inversion of **D** elements. The number of cycles remains lower than those required to compute the inverse of **D**. The proposed CIU generates the matrices $\mathbf{L}^{-1}$ and $\mathbf{D}^{-1}$ in $(i \cdot d + r)$ cycles for the ith iteration, where d is the latency of a multiply–subtract block and r the latency of the reciprocal circuit.

3) Inversion of Matrix **D***:* A diversity of division algorithms and their implementations has been published. Most of these algorithms are iterative and can be grouped into algorithms with linear convergence, such as CORDIC and restoring and nonrestoring division, and algorithms with quadratic convergence, such as reciprocal. To invert the elements of matrix **D**, we have developed a reciprocal operator based on the Newton–Raphson method to calculate the inverse of the diagonal elements D. This method consists of finding a function $f(X)$ that has a zero at $X = 1/M$. In general, this function is

$$f(X) = 1/X - M \tag{17}$$

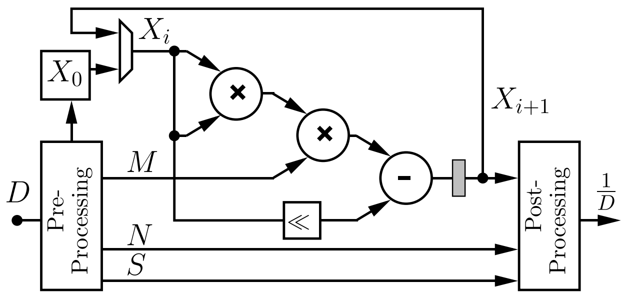


Fig. 6. Developed architecture of reciprocal operator PE$_d$.

and the Newton–Raphson iteration gives

$$X_{i+1} = X_i - \frac{f(X_i)}{f'(X_i)} = X_i(2 - MX_i) \tag{18}$$

which can be calculated from X_i using only multiplication and subtraction.

The Newton–Raphson method requires the initial start value to compute the solution approximation iteratively. Lookup table solutions for the initial value approximation are common, but by nature, they need quite large memories if high accuracy combined with low iteration counts is required. Alternatively, the initial value can be computed using a second-order polynomial, or linear approximation. The approximate value of $X_0 = (48 - 32M)/17$ is assumed for $1 \leq M < 2$ (or $0.5 < X \leq 1$) [20].

Fig. 6 shows a detailed structure of the developed reciprocal operator. The input data D can be either fixed-point or floating-point value. This value is preprocessed to extract sign S, the exponent N, and mantissa M ($M \in [1, 2]$). The initial value X_0 is computed using the approximation $X_0 = 3 - 2M$. We have replaced the numbers 48/17 and 32/17, respectively, by 3 and 2 to avoid the multiplication during preprocessing stage, thus only a subtraction and a right shift are used to compute X_0. Once X_0 is computed, the Newton–Raphson iteration is started. The result of approximation is postprocessed to obtain the final inverse $\mathbf{1/D}$. A controller (not shown in Fig. 6) schedules the operations of this PEs, particularly the number of iterations, which has an impact on the precision of the approximated value of the inverse $\mathbf{1/D}$. We have inserted pipeline at various levels of the proposed inversion design and studied their impact on latency, critical path, and resource utilization. We have also computed the time necessary to perform the inverse and found that the smallest is obtained for a latency of tree cycles and a maximum frequency of 92 MHz. The reciprocal element can operate at higher frequencies but with different latencies.

4) Inversion of Matrix $\mathbf{C}$*:* According to (8), $\mathbf{L}^{-1}$ and $\mathbf{D}^{-1}$ are multiplied to generate an intermediate symmetric matrix $\mathbf{Z} = \mathbf{D}^{-1}\mathbf{L}^{-1}$ that is further multiplied with transpose matrix $(\mathbf{L}^{-1})^T$ to generate the inverse matrix $\mathbf{C}^{-1} = (\mathbf{L}^{-1})^T\mathbf{Z}$, which is also symmetric. As $\mathbf{C}^{-1}$ is symmetric, only its upper triangular is calculated. According to (19), shown at the bottom of the page, the elements of matrix $\mathbf{C}^{-1}$ are updated in each iteration. The update consists of adding a new value, obtained by multiplying three elements of matrices $\mathbf{L}^{-1}$ and $\mathbf{D}^{-1}$, to the elements of $\mathbf{C}^{-1}$ computed in the previous iteration. This operation requires one addition and two multiplications, which are implemented by the proposed PE shown in Fig. 5.

5) Organization of CIU: The proposed structure for Cholesky decomposition consists of a scalable linear array of PEs with a complexity of $\mathcal{O}(m)$ number of PEs for an $m \times m$ matrix. An in-depth analysis of data dependency has demonstrated that an important number of operations can be performed in parallel (all operations in the same column of Fig. 4). This analysis led us to choose a structure of cascaded identical PEs (Fig. 5) organized to exploit all the options for parallelism (Fig. 7). The PEs are also pipelined to reduce the critical path. The proposed cascaded structure computes the elements of matrices $\mathbf{L}$ and $\mathbf{D}$ in $i \times d$ cycles in the ith iteration, where d is the latency of a multiply-and-subtract block (for the rest of this paper, we will call it a DSP cycle). These PEs are also utilized for the computation of $\mathbf{L}^{-1}$ and $\mathbf{C}^{-1}$.

The proposed structure is also composed of a division (PE$_d$) computing the inverse of diagonal elements of matrix $\mathbf{D}$. The activity of a PE depends on its position in the pipeline. For example, PE3 will be active from the third to the mth iteration and will take two DSP cycles in each iteration. This means that PEi will take $i - 1$ DSP cycles and will be active starting from the ith iteration. If we consider a sequential execution for m iterations, the total DSP cycles will be $(2m^3 + 6m^2 - 2m - 6)/6$, which requires computations of cubic order, $\mathcal{O}(m^3)$. The proposed linear array of PE takes $(m^2 + m - 2)/2$ DSP cycles for m iterations, which reduces the computations to a square order $\mathcal{O}(m^2)$ with a hardware cost of $\mathcal{O}(m)$. PE$_d$ computes the division of one element of

$$\begin{array}{lllllll}
\text{Iterations} & & 1 & 2 & 3 & 4 & 5 \\
C_{11}^{-1} & = & D_{11}^{-1} & + D_{11}^{-1}.L_{21}^{-1}.L_{21}^{-1} & + D_{11}^{-1}.L_{31}^{-1}.L_{31}^{-1} & + D_{11}^{-1}.L_{41}^{-1}.L_{41}^{-1} & + D_{11}^{-1}.L_{51}^{-1}.L_{51}^{-1} \\
C_{21}^{-1} & = & & + D_{11}^{-1}.L_{21}^{-1} & + D_{11}^{-1}.L_{31}^{-1}.L_{32}^{-1} & + D_{11}^{-1}.L_{41}^{-1}.L_{42}^{-1} & + D_{11}^{-1}.L_{51}^{-1}.L_{52}^{-1} \\
C_{22}^{-1} & = & & + D_{22}^{-1} & + D_{22}^{-1}.L_{32}^{-1}.L_{32}^{-1} & + D_{22}^{-1}.L_{42}^{-1}.L_{42}^{-1} & + D_{22}^{-1}.L_{52}^{-1}.L_{52}^{-1} \\
C_{31}^{-1} & = & & & + D_{11}^{-1}.L_{31}^{-1} & + D_{11}^{-1}.L_{41}^{-1}.L_{43}^{-1} & + D_{11}^{-1}.L_{51}^{-1}.L_{53}^{-1} \\
C_{32}^{-1} & = & & & + D_{22}^{-1}.L_{32}^{-1} & + D_{22}^{-1}.L_{42}^{-1}.L_{43}^{-1} & + D_{22}^{-1}.L_{52}^{-1}.L_{53}^{-1} \\
C_{33}^{-1} & = & & & + D_{33}^{-1} & + D_{33}^{-1}.L_{43}^{-1}.L_{43}^{-1} & + D_{33}^{-1}.L_{53}^{-1}.L_{53}^{-1}.
\end{array}
\begin{array}{r} \\ (19a) \\ (19b) \\ (19c) \\ (19d) \\ (19e) \\ (19f) \end{array}$$

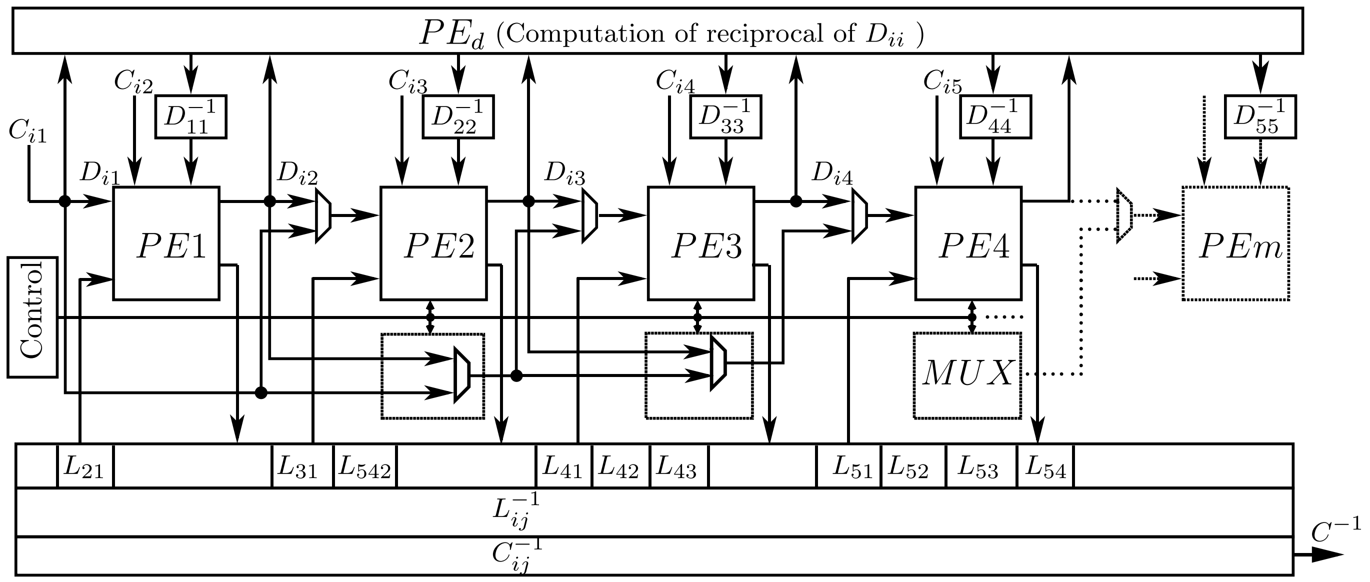


Fig. 7. Optimized and scalable linear array processors for matrix inversion based on Cholesky decomposition.

matrix **D** in each iteration, and will take the same number of cycles.

The proposed structure for matrix inversion is a regular and scalable hardware architecture. Due to its modularity, the size of matrix inversion can be changed easily. The scalability of the proposed architecture is closely related to its hardware complexity. The logic (PE and MUXs) and memory elements complexities are $O(m)$ and $O(m^2)$, respectively. More details on hardware and memory complexities are given in Tables III and IV.

C. Organization of the Proposed Architecture

According to (5), two matrix–vector multiplications are performed to compute an estimate of the original signal $\tilde{\mathbf{x}}$. In the first instance, transpose matrix $\tilde{\mathbf{\Phi}}^T$ is multiplied with residual vector $\mathbf{r}_{i-1}$ to generate an intermediate vector **W**. In the second instance, the inverse matrix $\mathbf{C}^{-1}$ is multiplied with **W** to generate the estimate vector $\tilde{\mathbf{x}}$. The K-IPCU described previously is used to calculate the first matrix–vector multiplication. The K-IPCU receives rows of $\tilde{\mathbf{\Phi}}^T$ and the residual vector $\mathbf{r}_{i-1}$ from the memory and performs IPC. This operation can be performed in parallel with the functions 1 and 2 of the architecture, and the values are saved in the memory unit. One of the four 64-point IP blocks of K-IPCU is used to perform the second matrix–vector multiplication. The rows of $\mathbf{C}^{-1}$ and intermediate vector $\mathbf{r}_{i-1}$ are retrieved from the memory and sent to K-IPCU to perform IPC. The estimate vector $\tilde{x}$ is saved in the memory unit to calculate the residual vector for the next iteration. The estimate vector $\tilde{\mathbf{x}}$ is multiplied with the generator matrix $\tilde{\mathbf{\Phi}}^T$, and then subtracted from the current measurement vector to find the residual vector for the next iteration. Multiplication of $\tilde{\mathbf{\Phi}}^T$ with vector $\tilde{\mathbf{x}}$ is performed by K-IPCU. It receives the rows of $\tilde{\mathbf{\Phi}}^T$ and the vector $\tilde{\mathbf{x}}$ from the memory and compute the IP. The K-IPCU completes the matrix–vector multiplication in K cycles and generates an intermediate vector.

IV. FPGA Implementation and Performance Evaluation

In this section, we present the implementation and performance evaluation of the proposed architecture, and the results in the case of $N = 1024$, $K = 256$, and $m = 36$.

A. Design Flow and FPGA Implementation Approach

We have used MATLAB–Simulink tool along with Xilinx system generator (XSG) and Xilinx LogiCore for the system-level modeling and representation of the proposed architecture for FPGA implementation of the OMP algorithm [21]. Using these toolset, we have transformed the abstract representation of our system-level design to a high-level description language (HDL) code for gate-level representation. XSG is a high-level design tool for Xilinx's line of FPGAs used as a plug-in to MATLAB–Simulink. It consists of a set of configurable computing blocks ranging from the basic combinational and sequential logic blocks to more complex math functions and control logic, along with a set of tools, such as system generator, resources estimator, and ModelSim, to realize the necessary functionalities. XSG supports both modular and hierarchical models of system representation that simplifies the design of the complex system.

We have described the proposed architecture in Simulink by block diagrams using the toolboxes available in Simulink library. Since we model the system for fixed-point implementation, we have provided the fixed-point format for each of the input blocks in Simulink description by specifying the number of bits and binary point in the properties of the block. We have also specified the parameters such as latency, and implementation options (to optimize for speed or area, and use of resources, such as embedded multipliers or DSP block set) for each of the building blocks by modifying their properties. The model of the system developed in XSG is simulated in MATLAB using a MATLAB script and generic Simulink blocks to create efficient test bench for simulation, test, and verification. The test bench defined in MATLAB–Simulink consists of the measurement matrix, measurement vector, and control signals, such as reset, enable, memory read/write, and steering logic. Data and signals are transferred to XSG–Simulink model through input gateways, and the reconstructed signal is transferred to the MATLAB environment for the visualization through output gateways. The simulation results obtained in MATLAB environment are bit-accurate and cycle-accurate and represent those which will be seen in the actual FPGA implementation. We have performed system simulation in Simulink and verified the functionality of the overall system.

We have run the HDL generation process of XSG that automatically compiles our design into low-level representations by invoking Xilinx CoreGen) to generate highly optimized netlists for the DSP building blocks. A close examination of the generated files shows that the HDL description is mainly structural, and a Xilinx native generic circuit file is almost associated to all entities.

In the final step of the design flow, we have performed validation followed by power and timing analysis. The project files generated by XSG are used by the project navigator tool of Xilinx Integrated Software Environment (ISE14.1). We verify the correctness of generated code by HDL simulation of test bench by utilizing ISE tool suite and ModelSim. We have created a constraint file for timing and power analysis of our design.

B. Validation and Evaluation Setup

The proposed architecture shown in Fig. 2 has been prototyped for $N = 1024$, $K = 256$, and $m = 36$ using the Simulink-based design flow discussed in Section IV-A. The model of the proposed architecture is designed using fixed-point data representation. The precision (number of bits) and the scale (binary point) of the fixed point data are defined as parameters. The system generator model of OMP architecture is configurable through model properties. In the model properties, model callbacks are used to initialize several functions, such as pipeline stages, and data precision parameter. To validate the proposed hardware implementation, and to compare software and hardware reconstruction efficiency, we have developed a MATLAB code of the OMP algorithm.

C. Reconstruction Efficiency

To evaluate the construction efficiency of the hardware approach, we have opted for an objective evaluation metric using peak signal-to-noise ratio (PSNR) defined as

$$\mathrm{PSNR} = 20\log_{10}\left(\frac{\mathrm{MAX}}{\sqrt{\mathrm{MSE}}}\right) \tag{20}$$

$$\mathrm{MSE} = \frac{1}{N} \times \sum_{i} [x(i) - \tilde{x}(i)]^2 \tag{21}$$

where MAX is the maximum possible value of the signal x, N is the total number of samples, $\tilde{x}(i)$ is the sample value at point i in the reconstructed signal, and $x(i)$ is the sample value at point i in the original signal. The average support-cardinality error (ASCE) metric gives valuable information on support-set distortion [22]. ASCE is defined as follows:

$$\mathrm{ASCE} = 1 - \frac{\#(\tilde{\Lambda} \cap \Lambda)}{m} \tag{22}$$

where m is the sparsity, and Λ and $\tilde{\Lambda}$ are, respectively, the index set (or support set) of the original signal $\mathbf{x}$ and the reconstructed signal $\tilde{\mathbf{x}}$. For our experiment, the original signal is represented in fixed-point representation and varies between -1 and $+1$. We had conducted simulations for data format $n(.f)$ with n the total number of bits (precision) and f the number of fractional bits. The obtained results are presented in Table II. ASCE = 0 corresponds to the cases where there is no distortion between the reconstructed signal and the original signal. In these cases, we observe a small increase in the quality of reconstruction of 2.4 dB when data precision is changed from 16(0.8) to 24(0.12) bits and 0.1 dB when data precision is changed from 24(0.12) to 32(0.16) bits. Compared with a similar architecture based on QR decomposition with the same data format [14], the quality of reconstruction using our architecture is superior by 10 dB. This is due to the fact that the design of [14] is based on MP algorithm, whereas the proposed design is based on the OMP algorithm. Note that, in the MP algorithm, the LSP is solved once after all iterations have been completed, whereas in the OMP algorithm, the LSP is solved in each iteration leading to a more precise estimate of reconstructed signal, obviously at the expense of computation complexity.

TABLE II
INFLUENCE OF DATA PRECISION ON QUALITY OF SIGNAL RECONSTRUCTION FOR $N = 1024$, $K = 256$, AND $m = 36$

Precision	**16 bits**		**24 bits**		**32 bits**	
Fractional	PSNR	ASCE	PSNR	ASCE	PSNR	ASCE
4	1	0.4	1	0.4	1	0.4
6	35.2	0	35.2	0	35.2	0
8	38.8	0	38.8	0	38.8	0
10	17	0.4	40.8	0	40.8	0
12	NC	NC	41.2	0	41.3	0
14	NC	NC	41.2	0	41.3	0
16	NC	NC	41.2	0	41.3	0
18	NC	NC	17	0,4	41.3	0
20	NC	NC	NC	NC	41.3	0
22	NC	NC	NC	NC	41.3	0
24	NC	NC	NC	NC	17	0,4

NC: PSNR and ASCE are not calculated when the signal is not completely reconstructed.

D. Hardware Complexity

The hardware complexity of the proposed architecture is evaluated and listed in Table III. It involves $(K + 3m)$ number of multipliers, $(K + 2m)$ number of adder/subtractors, and one reciprocal unit. In the proposed design, only one reciprocal is used for the inversion of elements of matrix $\mathbf{D}$.

Table IV shows the type, the size, and the number of accesses per cycle in read and writes modes of memory

TABLE III
THEORETICAL ESTIMATE OF HARDWARE COMPLEXITY OF THE PROPOSED DESIGN

		Multiplier	Adder/substractor	Comparator	Negate	Reciprocal
$\mathbf{K-IPCU}$		K	$K-1$	1	0	0
$\mathbf{RCU}$		0	1	0	0	0
CIU	$\mathbf{L}$	0	0	0	0	0
	$\mathbf{D^{-1}}$	0	0	0	0	0
	$\mathbf{L^{-1}}$	0	0	0	m	0
	$\mathbf{C^{-1}}$	$2m$	m	0	0	1
$\mathbf{XCU}$		m	m	0	0	0

TABLE IV
THEORETICAL ESTIMATE OF MEMORY RESOURCES

Memory Elements		$\mathbf{y}$	$\mathbf{r}$	$\mathbf{\Phi}$	$\mathbf{C}$	$\mathbf{w}$	$\mathbf{L}$	$\mathbf{D^{-1}}$	$\mathbf{L^{-1}}$	$\mathbf{C^{-1}}$	$\tilde{\mathbf{x}}$
Type		BRAM	BRAM	BRAM	Reg	Reg	Reg	Reg	Reg	Reg	Reg
Size		K	K	KN	$m(m+1)/2$	m	$m(m-1)/2$	m	$m(m-1)/2$	$m(m-1)/2$	$2m$
Word per cycle	R	1	K	K	m	m	$m-1$	1	$m(m-1)/2$	$m(m-1)/2$	$2m$
	W	1	1	†	m	1	$m-1$	m	$m-1$	1	$2m$

LEGEND. Reg: Registers; BRAM: bloc RAM; † : Matrix Φ is a constant

resources. Bloc RAMs (BRAMs) are utilized to store vectors $\mathbf{y}$ and $\mathbf{r}$ and matrix $\mathbf{\Phi}$. The remaining memory element is implemented in registers when it is necessary to exploit the possible parallelism of different calculations. BRAM is also organized for sequential or parallel access. For example, vector $\mathbf{y}$ is accessed sequentially in read and write modes, while register $\mathbf{r}$ is written in sequential mode and read in parallel as the IP $\langle \mathbf{r_{i-1}}, \phi_\mathbf{j} \rangle$ is computed in parallel. Thus, the K elements of vector r_{i-1} and $\phi_\mathbf{j}$ should be read in one clock cycle. Therefore, the memory bandwidth, expressed here in words per cycle, depends on data access mode (read/write) and the quantity of elements required per cycle.

E. Implementation Results and Timing Analysis

For this paper, we targeted the ML605 board equipped with the Virtex6 FPGA (part xc6vlx240t-1ff1156). The algorithm is prototyped using system generator in which several parameters are modified allowing the architecture exploration. Among these parameters, we explored the impact of data precision and the latency of pipelining. For the comparison purposes with similar work, we have used 18-bit data precision with 9 bits for the fractional part, which allows the reconstruction of the signal with a PSNR of 38.9 dB and an ASCE of 0.

The reconstruction time is the product of minimum cycle period and the number of cycles necessary to reconstruct a vector data. The K-IPCU and the comparator have eight pipeline stages and computes N IPs and comparison in $m(N + 8)$ cycles for m iterations, which corresponds to 37 152 cycles for $N = 1024$ and $m = 36$. The elements of matrix $\mathbf{C}$ are computed by a part of K-IPCU, as explained in Section III-A. The number of cycles depends on the iteration, and the total is $(m^2 + 13m)/2$. The latency of the CIU also depends on the iteration; it takes 5 cycles in the first iteration and $3i + 5$ cycles in the ith iteration. This variation is due to Cholesky decomposition that requires extra 3 cycles in each iteration because of the growing size of matrix C, as explained above. The proposed architecture for matrix inversion based on the modified Cholesky decomposition takes $(3m^2+13m)/2$ cycles to invert an $m \times m$ matrix, which corresponds to 2178 cycles in our case. The latency for the computation of $\tilde{x}$ is constant for each iteration and takes $3m$. The total number of cycles for the reconstruction of signal of the proposed architecture is $2m^2+(N+37)m$, which corresponds to 40 788 for the configuration ($N = 1024$, $K = 256$, and $m = 36$).

In the proposed architecture, the inversion is based on a reciprocal IP core that allows significant reduction of latency. Moreover, we have modified the controller to start the computation of inverse of matrix $\mathbf{L}$ as soon as its elements are available. To reduce the critical path, we have optimized architecture by inserting pipeline registers for different embedded multipliers and for reciprocal IP core.

F. Comparisons With Existing Approaches

We have compared the performance of proposed design with the existing similar designs proposed for the OMP algorithm [17]. We have implemented the proposed design for $N = 1024$, $K = 256$, and $m = 36$ using Virtex6 FPGA device. We have used DSP48 offered by Virtex6 FPGA for the implementation of multiplication–add/multiplication–subtract operation. Similarly, we have used RAM blocks (RAMBs) of Virtex6 FPGA for the implementation of all buffers required by the proposed design for storing intermediate matrices and vectors. We have considered 18-bit precision for all the intermediate and output signals. The device utilization summary and timing obtained from place and route report are listed in Table V. The synthesis results of the existing design, as

TABLE V
COMPARISON OF IMPLEMENTATION RESULTS OF THE PROPOSED ARCHITECTURE AND THE EXISTING DESIGN FOR $N = 1024$, $K = 256$, AND $m = 36$ USING VIRTEX6 FPGA FROM XILINX, AND 18 DATA PRECISION

Design	**[17]** Virtex6	**Proposed** Virtex6
Max Frequency (MHz)	100	119.96
Occupied Slices	32010(84%)	6208(16%)
DSP Core	261(33%)	589(76%)
RAMB	258(62%)‡	576(69%)†
Reconstruction Time (μs)	630	340
Dynamic power (mW)	N/A	3233

† The type of RAM blocks is RAMB18E1, which stores up to $18K$ bits.
‡ The type of RAM block is not indicated in [17].

reported in [17] for $N = 1024$ and $K = 256$, are also listed in Table V for comparison. Compared with [17], the proposed structure involves 328 more DSP48s and 318 more RAMBs, but it involves 25 802 less slices and 1.85 times less computation time for signal reconstruction. The proposed structure offers 15 dB higher PSNR for signal reconstruction, which is very significant for compressing sensed date. In addition, the proposed structure is fully scalable for different size of OMP algorithm and can be used for reconstruction of signals for unknown sparsity. Most importantly, the proposed design can be easily prototyped for resource constrained FPGA device for better quality signal reconstruction by appropriate selection of bit width.

V. CONCLUSION

We have presented a scheme for FPGA implementation of the OMP algorithm for reconstruction of compressively sensed signal. We have modeled the proposed design using Simulink for the implementation on FPGA using XSG tool. In addition, we have presented a methodology to optimize both area and execution time. The execution time is reduced by exploiting parallelism inside each of the dependent functions, and the area consumption is reduced by reusing the hardware of matrix–vector multiplication for other components of the algorithm. We have evaluated the quality of data reconstruction of our architecture using PSNR and ASCE metrics for different data precisions. We have shown that our proposed architecture provides a higher PSNR of 38.9 dB, which is superior by 15 dB compared with a recently reported reconstruction architecture having similar parameters ($N = 1024$, $K = 256$, and $m = 36$) with data precision of 18 and 9 bits for fractional. The proposed structure involves 328 more DSP48s, but it involves 25 802 less slices and 1.85 times less computation time for signal reconstruction. The proposed design is fully scalable for higher sparsity and can be adapted for unknown sparsities. In addition, we have presented a complete high-level design flow for rapid prototyping of OMP algorithm in resource constrained FPGA for better quality signal reconstruction.

REFERENCES

[1] E. J. Candes and M. B. Wakin, “An introduction to compressive sampling,” *IEEE Signal Process. Mag.*, vol. 25, no. 2, pp. 21–30, Mar. 2008. [Online]. Available: http://dx.doi.org/10.1109/MSP.2007.914731

[2] A. M. R. Dixon, E. G. Allstot, A. Y. Chen, D. Gangopadhyay, and D. J. Allstot, “Compressed sensing reconstruction: Comparative study with applications to ECG bio-signals,” in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2011, pp. 805–808.

[3] D. Yang, H. Li, G. D. Peterson, and A. Fathy, “Compressed sensing based UWB receiver: Hardware compressing and FPGA reconstruction,” in *Proc. 43rd Annu. Conf. Inf. Sci. Syst. (CISS)*, Mar. 2009, pp. 198–201.

[4] S. G. Mallat and Z. Zhang, “Matching pursuits with time-frequency dictionaries,” *IEEE Trans. Signal Process.*, vol. 41, no. 12, pp. 3397–3415, Dec. 1993.

[5] P. Dymarski, N. Moreau, and G. Richard, “Greedy sparse decompositions: A comparative study,” *EURASIP J. Adv. Signal Process.*, vol. 2011, p. 34, Aug. 2011.

[6] P. Maechler *et al.*, “VLSI design of approximate message passing for signal restoration and compressive sensing,” *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 2, no. 3, pp. 579–590, Sep. 2012.

[7] J. A. Tropp and A. C. Gilbert, “Signal recovery from random measurements via orthogonal matching pursuit,” *IEEE Trans. Inf. Theory*, vol. 53, no. 12, pp. 4655–4666, Dec. 2007.

[8] T. R. Braun, “An evaluation of GPU acceleration for sparse reconstruction,” *Proc. SPIE, Signal Process., Sensor Fusion, Target Recognit. XIX*, vol. 7697, no. 15, pp. 769715-1–769715-10, Apr. 2010.

[9] M. Andrecut, “Fast GPU implementation of sparse signal recovery from random projections,” *Eng. Lett.*, vol. 17, no. 3, pp. 151–158, 2009.

[10] O. Maslennikow, V. Lepekha, A. Sergiyenko, A. Tomas, and R. Wyrzykowski, “Parallel implementation of Cholesky LL^T-algorithm in FPGA-based processor,” in *Proc. 7th Int. Conf. Parallel Process. Appl. Math.*, 2008, pp. 137–147. [Online]. Available: http://dl.acm.org/citation.cfm?id=1786194.1786211

[11] A. Happonen, A. Burian, and E. Hemming, “A reconfigurable processing element implementation for matrix inversion using Cholesky decomposition,” *World Acad. Sci., Eng., Technol.*, vol. 3, no. 28, pp. 114–117, Mar. 2004.

[12] D. Yang, G. D. Peterson, and H. Li, “High performance reconfigurable computing for Cholesky decomposition,” in *Proc. Symp. Appl. Accel. High Perform. Comput. (UIUC)*, Jul. 2009.

[13] A. Septimus and R. Steinberg, “Compressive sampling hardware reconstruction,” in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May/Jun. 2010, pp. 3316–3319.

[14] J. L. V. M. Stanislaus and T. Mohsenin, “High performance compressive sensing reconstruction hardware with QRD process,” in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2012, pp. 29–32.

[15] J. L. V. M. Stanislaus and T. Mohsenin, “Low-complexity FPGA implementation of compressive sensing reconstruction,” in *Proc. Int. Conf. Comput., Netw., Commun. (ICNC)*, Jan. 2013, pp. 671–675.

[16] P. Blaché, H. Rabah, and A. Amira, “High level prototyping and FPGA implementation of the orthogonal matching pursuit algorithm,” in *Proc. 11th Int. Conf. Inf. Sci., Signal Process., Appl. (ISSPA)*, Jul. 2012, pp. 1336–1340.

[17] L. Bai, P. Maechler, M. Muehlberghuber, and H. Kaeslin, “High-speed compressed sensing reconstruction on FPGA using OMP and AMP,” in *Proc. 19th IEEE Int. Conf. Electron., Circuits, Syst. (ICECS)*, Dec. 2012, pp. 53–56.

[18] P. E. Gill and W. Murray, “Newton-type methods for unconstrained and linearly constrained optimization,” *Math. Program.*, vol. 7, no. 1, pp. 311–350, 1974. [Online]. Available: http://dx.doi.org/10.1007/BF01585529

[19] (Jan. 2012). *Virtex-5 FPGA XtremeDSP Design Considerations*. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug193.pdf

[20] *Division Algorithm*. [Online]. Available: http://en.wikipedia.org/wiki/Division_algorithm, accessed Apr. 2013.

[21] (Apr. 2012). *System Generator for DSP*. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/sysgen_user.pdf, accessed Apr. 2013.

[22] S. Chatterjee, D. Sundman, M. Vehkaperä, and M. Skolglund, “Projection-based and look-ahead strategies for atom selection,” *IEEE Trans. Signal Process.*, vol. 60, no. 2, pp. 634–647, Feb. 2012.

Hassan Rabah (M'06–SM'13) received the M.S. degree in electronics and control engineering and the Ph.D. degree in electronics from Henri Poincaré University, Nancy, France, in 1987 and 1993, respectively.

He became an Associate Professor of Electronics Microelectronics and Reconfigurable Computing with the University of Lorraine, Nancy, in 1993, where he also became a Full Professor in 2011. In 1997, he joined the Architecture Group of the Laboratoire d'Instrumentation Electronique de Nancy, Nancy, France, where he supervised research on VLSI implementation of parallel architecture for image and video processing. He also supervised research in the field-programmable gate array (FPGA) implementation of adaptive architectures for smart sensors in collaboration with industrial partners. He participated in several national projects for quality of service measurement and video transcoding techniques. In 2013, he joined the Institut Jean Lamour at the University of Lorraine, where he is currently the Head of the Measurement and Electronic Architectures Group. His current research interests include partial and dynamic reconfigurable architectures for adaptive systems, design, implementation of FPGA-based embedded systems with an emphasis on power optimization, video compression decompression and transcoding, compressive sensing, and sensor networks.

Dr. Rabah has been the Program Committee Member for a number of conferences.

Abbes Amira (M'01–SM'07) received the Ph.D. degree in computer engineering from Queen's University Belfast, Belfast, U.K., in 2001, with a minor in developing a coprocessor for matrix algorithms using field-programmable gate arrays for image and signal processing applications.

He held academic and consultancy positions in U.K. and overseas, including the positions as a Full Professor of Visual Communications and a Leader of the Visual Communication Cluster with the University of the West of Scotland, Paisley, U.K., and a Professor of Computer Engineering with Qatar University, Doha, Qatar. He held other academic positions with the University of Ulster, Londonderry, U.K., Qatar University, Brunel University, Uxbridge, U.K., and Queen's University Belfast. He was a Visiting Professor with the University of Lorraine, Nancy, France. He holds a Visiting Professor position with the Tun Hussein Onn University of Malaysia, Johor Bahru, Malaysia. He is a regular referee for many national and international funding bodies, including the Engineering and Physical Research Council and the Qatar National Research Fund, and he sits on the international advisory boards for some international research centers. He has successfully supervised 13 Ph.D. students. He has authored over 200 publications in reconfigurable computing, and image and signal processing. His current research interests include embedded systems, high-performance reconfigurable computing, image and video processing, multiresolution analysis, biometrics, and connected health applications.

Dr. Amira is a fellow of the Institution of Engineering and Technology, and the Higher Education Academy, and a Senior Member of the Association for Computing Machinery. He has been awarded a number of grants from the government and industry. He has been invited to give talks, short courses, and tutorials at universities and international conferences, and is the Chair and Program Committee Member for a number of conferences. He was one of the tutorials presenters at the International Conference on Image Processing in 2009, the Chair of the Embedded Computer Vision Workshop (ECVW) in 2011, the Program Chair of ECVW in 2010, the Program Co-Chair of the International Conference on Microelectronics (ICM) in 2012, the DELTA Conference in 2008, the Irish Machine Vision and Image Processing Conference in 2005, and the General Co-Chair of ICM in 2014.

Basant Kumar Mohanty (M'06–SM'11) received the M.Sc. degree in physics from Sambalpur University, Sambalpur, India, in 1989 and the Ph.D. degree in VLSI for digital signal processing from Berhampur University, Brahmapur, India, in 2000.

He joined as a Lecturer with the Department of Electrical and Electronics Engineering, Birla Institute of Technology and Science, Pilani, India, in 2001, and then as an Assistant Professor with the Department of Department of Electronics and Communication Engineering, Mody Institute of Education and Research (Deemed University), Sikar, India. In 2003, he joined the Jaypee University of Engineering and Technology, Guna, India, where he became an Associate Professor in 2005 and a Full Professor in 2007. He has authored over 50 technical papers in reputed international journals and conferences. His current research interests include design and implementation of low-power and high-performance systems for adaptive filters, image and video processing applications, secured communication, and reconfigurable architectures.

Dr. Mohanty is a life time member of the Institution of Electronics and Telecommunication Engineering, New Delhi, India. He was a recipient of the Rashtriya Gaurav Award from the India International Friendship Society, New Delhi, in 2012. He serves as an Associate Editor of the *Journal of Circuits, Systems, and Signal Processing*.

Somaya Almaadeed (SM'12) received the Ph.D. degree in computer science from the University of Nottingham, Nottingham, U.K., in 2004.

She has been a Visiting Fellow with Northumbria University, Newcastle upon Tyne, U.K., since 2012. She is currently with the Department of Computer Science and Engineering, Qatar University, Doha, Qatar, as an Assistant Professor, where she is involved in research on biometrics, writer identification, image processing, and document analysis. She has been awarded a number of grants, and has authored around 40 papers.

Dr. Almaadeed is a member of different international computer science committees. Her team received the Best Performance Award in the 2011 International Conference on Document Analysis and Recognition's Signature Verification Competition and Music Scores Competition.

Pramod Kumar Meher (SM'03) received the B.Sc. (Hons.) and M.Sc. degrees in physics and the Ph.D. degree in science from Sambalpur University, Sambalpur, India, in 1976, 1978, and 1996, respectively.

He is currently a Senior Research Scientist with Nanyang Technological University, Singapore. He was a Professor of Computer Applications with Utkal University, Bhubaneswar, India, from 1997 to 2002, and a Reader in Electronics with Berhampur University, Brahmapur, India, from 1993 to 1997. His current research interests include design of dedicated and reconfigurable architectures for computation-intensive algorithms pertaining to signal, image and video processing, communication, bioinformatics, and intelligent computing. He has contributed over 200 technical papers to various reputed journals and conference proceedings.

Dr. Meher is a Fellow of the Institution of Electronics and Telecommunication Engineers in India. He was the recipient of the Samanta Chandrasekhar Award for excellence in research in Engineering and Technology in 1999. He has served as a speaker of the Distinguished Lecturer Program of the IEEE Circuits Systems Society from 2011 to 2012, and an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS from 2008 to 2011 and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS from 2012 to 2013. He currently serves as an Associate Editor of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS and the *Journal of Circuits, Systems, and Signal Processing*.