Received April 13, 2018, accepted May 28, 2018, date of publication May 30, 2018, date of current version June 20, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2842076

Kernel Solver Design of FPGA-Based Real-Time Simulator for Active Distribution Networks

ZHIYING WANG[1], FANPENG ZENG[2], PENG LI[1], CHENGSHAN WANG[1], (Senior Member, IEEE), XIAOPENG FU[1], (Member, IEEE), AND JIANZHONG WU[3], (Member, IEEE)

[1]Key Laboratory of Smart Grid of the Ministry of Education, Tianjin University, Tianjin 300072, China
[2]State Grid Tianjin Dongli Electric Power Supply Company, Tianjin 300300, China
[3]School of Engineering, Institute of Energy, Cardiff University, Cardiff CF24 3AA, U.K.

Corresponding author: Peng Li (lip@tju.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 51577127 and in part by the Technology Project of State Grid under Grant PD71-16-005.

ABSTRACT The field-programmable gate array (FPGA)-based real-time simulator takes advantage of many merits of FPGA, such as small time-step, high simulation precision, rich I/O interface resources, and low cost. The sparse linear equations formed by the node conductance matrix need to be solved repeatedly within each time-step, which introduces great challenges to the performance of the real-time simulator. In this paper, a fine-grained solver of the FPGA-based real-time simulator for active distribution networks is designed to meet the computational demand. The framework of the solver, offline process design on PC and online process design on FPGA are proposed in detail. The modified IEEE 33-node system with photovoltaics is simulated on a 4-FPGA-based real-time simulator. Simulation results are compared with PSCAD/EMTDC under the same conditions to validate the solver design.

INDEX TERMS Real-time simulation, field programmable gate array (FPGA), sparse matrix, LU decomposition, parallel scheduling, active distribution networks (ADN).

I. INTRODUCTION

Real-time (RT) simulator for active distribution networks (ADN) integrated with various renewable energy resources is capable of reproducing the dynamic behaviors of the real system being modeled in detail [1]–[3]. It makes possible for the testing and validation of the system equipment and operation strategies, such as distributed generator (DG) prototype and its controllers, system-level voltage and frequency regulation strategies, protective devices and so on [4]. An RT simulator is required to complete the computation of the model equations that characterize the system behavior at a pace of the real-world clock time [5]. For the real-time simulation of ADNs, the selected time-step should be tens of microseconds to track the high-frequency transients and thus preserve the fidelity of simulation [6]. However, as an essential process, solving the linear equations resulting from the model equations of ADNs with increasing large-scales is time-consuming and computational demanding [7]. It severely restricts the calculation speed of the simulator so that RT simulation can hardly be reached. A high-speed linear equation solver with sufficient accuracy is desired to meet the real-time constraints.

Due to the expanding scale of ADNs and fast dynamic characteristics brought by DG integration, solving the enlarged linear equations in real-time is challenging. The calculation speed and computation burden are two conflicts for RT simulators. The increasing scale of ADNs expand the dimension of the linear equations. The solving process of such equations involves massive arithmetic operations, which requires plenty of computation resources. Besides, the selected time-step for simulation is generally determined by the smallest time constant (usually relating to the power electronic devices). To meet the real-time constraint, more computation resources are needed to complete solving the equations.

The existing solutions for linear equations can be classified into two categories: direct methods and iterative methods [8]–[11]. The direct methods attempt to solve the equations by a finite sequence of operations that the unknown variables are eliminated successively. The iterative methods use an initial guess to generate a sequence of improving approximate solutions. The iteration will not stop until the corresponding sequence converges for the given initial approximations. For the linear problems involving a large number of variables (sometimes of the order of millions), the iterative methods are often useful, where the direct methods would be prohibitively expensive (and in some cases impossible) even with the powerful computing resources [12]–[14]. For the medium-scale linear systems, the direct methods are considered to be more efficient [12]–[14]. In addition, when using iterative linear solvers, the solution time per time-step depends on the number of iterations and this number can change from solution point to solution point. For the real-time applications, the computation hardware needs to be synchronized at evenly spaced time-steps with a global clock, thus the solution time per time-step should be constant, which is a conflict requirement. Considering that the scale of the studied power system is not very large and the real-time computing is difficult to be guaranteed by iterative methods, the direct methods are more popularly employed by the advanced RT simulators, such as RTDS®, RT-LAB® and HYPERSIM® [1]. Although a series of well-developed linear equation solutions have been proposed, in RT simulators built with CPU or DSP based sequential hardware, solving the linear equations is still a bottleneck due to the relatively slower calculation speed and limited computational power of such hardware.

The inherent parallel hardware, field programmable gate array (FPGA) has provided a feasible solution to meet the need for high-performance linear equation solvers [15]. The main advantages of FPGA is the massive parallelism, deep pipeline, and rich distributed memories [16]. A number of research efforts have been conducted to solve the linear equations on FPGAs [17]–[19]. In [17], a block LU decomposition algorithm using FPGA has been presented in which memory accesses were studied for various FPGA configurations. Reference [18] also proposed a block LU decomposition algorithm on FPGAs that was applicable for arbitrary matrix size. The computation time for the aforementioned approaches are on the order of hundreds of milliseconds or even tens of seconds. Reference [19] proposed a small-scale real-time nonlinear electromagnetic transient solver on FPGA, in which the partially linear algebraic equations were solved by Gauss-Jordan Elimination.

In this paper, a kernel solver design of FPGA-based real-time simulator for ADNs using direct method is proposed. In the direct methods set, LU decomposition algorithm [20] is one of the commonly used method to solve linear equations. It has been included in many popular linear algebra libraries such as Linear Algebra Package (LAPACK) [21] and Linear System Package (LINPACK) [22]. In this paper, the LU decomposition algorithm is also adopted to solve the linear equations for its simpler and easier hardware implementation. The main solving procedure consists of LU decomposition, forward substitution and backward substitution. Before the decomposition, the coefficient matrix of the linear equations should be rearranged (pivoting) since the original matrix may not support LU decomposition. The column approximate minimum degree (COLAMD) algorithm [23] is adopted to realize the matrix rearrangement. To save the computation time, topology analysis of the resulting upper and lower triangular matrix is essential to compact the column operations in the forward and backward substitution. It should be noted that the matrix rearrangement and LU decomposition is in essence a series of elementary transformations that is realized by matrix multiplication. It involves massive arithmetic operations and consumes plenty of time. Thus, the matrix rearrangement, LU decomposition and topology analysis are performed in offline. While the forward and backward substitutions are realized in online. As the two substitution are in the same iterative scheme, a unified hardware block is designed in the solver. When performing the substitutions on FPGA, the compressed sparse format [24] of the decomposed triangular matrices is adopted to save storage resources. To demonstrate the proposed FPGA-based solver, a modified IEEE-33 node system with photovoltaics (PVs) is simulated on a 4-FPGA-based real-time platform. The simulation results are compared with the offline commercial program PSCAD/EMTDC® to verify the correctness and effectiveness of the design.

The main contributions of this paper are summarized as follows: i) The framework of the FPGA-based real-time solver is built which divides the work into offline part and online part; ii) For the offline design, the detailed scheduling of the subtasks for the substitutions are presented in terms of the matrix arrangement and LU decomposition algorithm; iii) For the online design, the compressed sparse format for the decomposed triangular matrices and the column operations splitting for substitutions are realized. The hardware design of the FPGA-based solver is presented as well.

The structure of this paper is organized as follows. Section II introduces the framework of the FPGA-based real-time solver. Section III provides the detailed design of the solver, including the offline part on PC, the online part on FPGA and the hardware design of the solver. In Section IV, the modified IEEE 33-node system with PVs is simulated on a 4-FPGA-based real-time simulator. The performance of the solver is analyzed in detail as well. The conclusions are stated in Section V.

II. FRAMEWORK OF FPGA-BASED REAL-TIME SOLVER

In this paper, the modified augmented nodal analysis (MANA) framework is chosen to implement the real-time simulation of ADNs [25]. The framework basically consists of three steps: i) Discretizing the branch equation of each element in ADNs yields the difference equations; ii) Forming and solving the overall system equations to obtain the node voltages; iii) Updating the historical voltages and currents for each element.

In step ii), the overall system equations, known as nodal equation, can be established as (1)

$$\boldsymbol{Gu} = \boldsymbol{i} \tag{1}$$

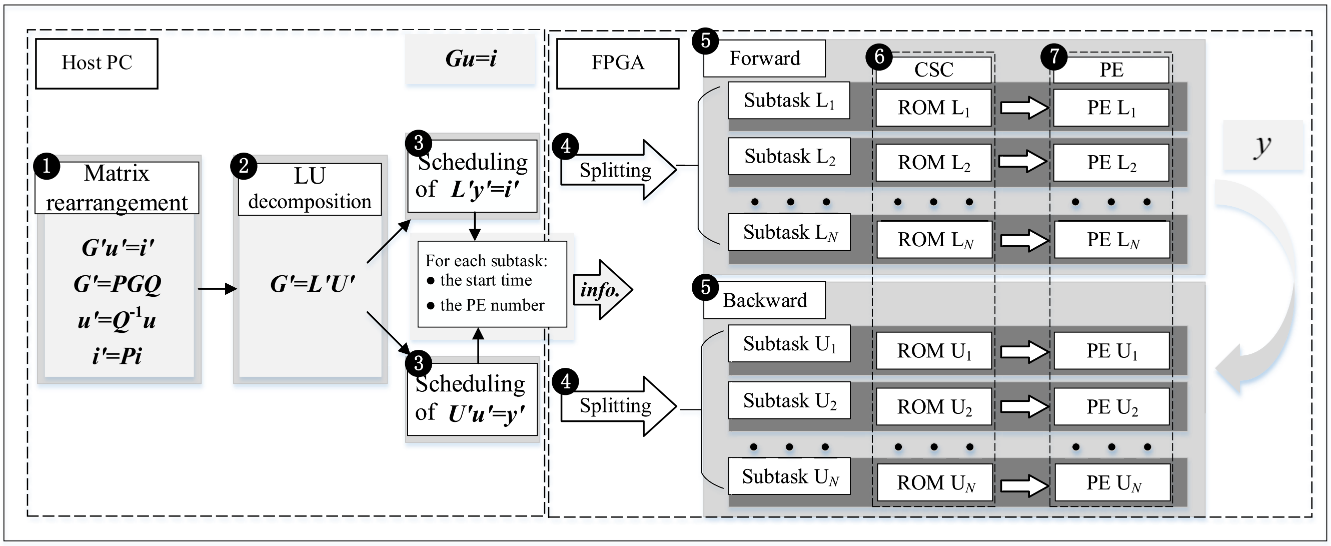


FIGURE 1. The framework of the FPGA-based real-time solver.

in which $\boldsymbol{G}$ denotes the nodal conductance matrix, $\boldsymbol{u}$ denotes the vector of node voltages, and $\boldsymbol{i}$ denotes the vector of node injected currents. Generally, the system matrix $\boldsymbol{G}$ resulting from MANA is not symmetric due to the ideal switch modelling [25]. In this paper, the switches are modelled as an alternative resistance for the sake of the numerical stability. Consequently, the system matrix $\boldsymbol{G}$ becomes a symmetric matrix.

The LU decomposition algorithm used to solve (1) is constructed by LU decomposition, forward substitution and backward substitution expressed by (2)-(4). With the premise that the ranks of the certain submatrices for $\boldsymbol{G}$ are nonzero, $\boldsymbol{G}$ can be decomposed into an upper triangular matrix $\boldsymbol{U}$ and a lower triangular matrix $\boldsymbol{L}$. Specially, the symmetry of the matrix $\boldsymbol{G}$ is exploited in the sense that the upper triangular matrix $\boldsymbol{U}$ is the transposed matrix of $\boldsymbol{L}$. After forward and backward substitutions successively, $\boldsymbol{u}$ will be obtained. $\boldsymbol{y}$ is the intermediate result. Note that the original $\boldsymbol{G}$ resulting from ADNs may not support LU decomposition, the elements of $\boldsymbol{G}$ have to be rearranged to meet the decomposition conditions.

$$\boldsymbol{G} = \boldsymbol{LU} \tag{2}$$
$$\boldsymbol{Ly} = \boldsymbol{i} \tag{3}$$
$$\boldsymbol{Uu} = \boldsymbol{y} \tag{4}$$

Fig. 1 shows the detailed framework of the real-time solver. It consists of the offline process on host PC and the online process on FPGA. In the offline process, the original $\boldsymbol{G}$, $\boldsymbol{u}$ and $\boldsymbol{i}$ are converted to $\boldsymbol{G}'$, $\boldsymbol{u}'$ and $\boldsymbol{i}'$ respectively by elementary transformation. The detailed transition will be described in Section III.A. Then the LU decomposition algorithm is employed to accomplish the numerical factorization of $\boldsymbol{G}'$, obtaining $\boldsymbol{L}'$ and $\boldsymbol{U}'$. Finally, scheduling the subtasks of the forward and backward substitutions according to the topology analysis of $\boldsymbol{L}'$ and $\boldsymbol{U}'$ is performed to generate the scheduling datasheet. The datasheet, $\boldsymbol{L}'$ and $\boldsymbol{U}'$ are downloaded into FPGA. In the online process, the elements of $\boldsymbol{L}'$ and $\boldsymbol{U}'$ are stored in the compressed sparse format, which is in favor of storage cost reduce and dataflow deep pipelines. According to the scheduling datasheet, the entire solving task for each substitution is split into N subtasks. N is the dimension of $\boldsymbol{L}'$ and $\boldsymbol{U}'$. Each subtask is completed by a column operation using a processing element (PE) which will be described in Section III.C in detail.

III. DETAILED DESIGN OF THE REAL-TIME SOLVER

In this section, the detailed design of the solver based on FPGA is promoted. The implementation details of both the offline and online design are provided. The hardware design of the solver is presented as well.

A. OFFLINE DESIGN

The offline design on PC accomplishes the rearrangement of the matrix and finds the scheduling order of the subtasks for forward and backward substitutions. It determines the parallelism and pipeline depth of the online design on FPGA.

1) MATRIX REARRANGEMENT

To ensure that the LU decomposition is validate and the resulting $\boldsymbol{L}$ and $\boldsymbol{U}$ have better sparsity, COLAMD algorithm is employed for sparse partial pivoting of $\boldsymbol{G}$. It provides a sparsity preserving column pre-ordering prior to numerical factorization. The detailed procedure is as follows:

i) Find a column permutation matrix $\boldsymbol{Q}$ which is selected without regard to the numerical values;

ii) Find the row permutation matrix $\boldsymbol{P}$ via standard partial pivoting without regard to sparsity;

The equation of (1) for solving can be rewritten as (5)

$$\boldsymbol{G}'\boldsymbol{u}' = \mathbf{i}' \tag{5}$$

where $\boldsymbol{G}'$, $\boldsymbol{u}'$ and $\boldsymbol{i}'$ are obtained by

$$\boldsymbol{G}' = \boldsymbol{PGQ} \tag{6}$$
$$\boldsymbol{u}' = \boldsymbol{Q}^{-1}\boldsymbol{U} \tag{7}$$
$$\boldsymbol{i}' = \boldsymbol{Pi} \tag{8}$$

The forward and backward substitutions are expressed as

$$\boldsymbol{L}'\boldsymbol{y}' = \boldsymbol{i}' \tag{9}$$
$$\boldsymbol{U}'\boldsymbol{u}' = \boldsymbol{y}' \tag{10}$$

in which $\boldsymbol{y}'$ is the updated intermediate result. It is noted that the matrix $\boldsymbol{Q}$ should be found to limit the worst-case fill-in, regardless of how $\boldsymbol{P}$ is subsequently chosen.

As $\boldsymbol{P}$ and $\boldsymbol{Q}$ are permutation matrices which only change the location of the elements in $\boldsymbol{u}$ and $\boldsymbol{i}$, the calculation of (7) and (8) are realized by recording the storage addresses of $\boldsymbol{u}$ & $\boldsymbol{i}$ and $\boldsymbol{u}'$ & $\boldsymbol{i}'$ to avoid the matrix operation.

2) LU DECOMPOSITION ALGORITHM

A left-looking LU decomposition algorithm [26] is adopted to realize the numerical factorization of $\boldsymbol{G}'$. The entire algorithm is described in Algorithm 1. Especially, $\boldsymbol{U}'$ is normalized (line 19) to force the diagonal elements of $\boldsymbol{U}'$ to **1** as $\boldsymbol{L}'$ does. Thus the forward and backward substitutions will have the same format and the hardware design of the PE will be simplified. The execution of each substitution includes N subtasks. During the subtask, the elements of a certain column of $\boldsymbol{L}'$ and $\boldsymbol{U}'$ are updated by (9) and (10).

Algorithm 1 Left-Looking Decomposition of a n-by-n G'

```
1:  for $j = 1$ to $n$ do
2:    for $i = 1$ to $j - 1$ where $G'(i, j) \neq 0$ do
3:      for $k = i + 1$ to $n$ where $G'(k, i) \neq 0$ do
4:        $G'(k, j) = G'(k, j) - G'(k, i) * G'(i, j)$;
5:      end for
6:    end for
7:    for $k = j + 1$ to $n$ where $G'(k, j) \neq 0$ do
8:        $G'(k, j) = G'(k, j)/G'(j, j)$;
9:  end for
10: end for
11: for $i = 2$ to $n$ do
12:   for $j = 1$ to $j - 1$ where $G'(i, j) \neq 0$ do
13:     $L'(i, j) = G'(i, j)$;
14:   end for
15: end for
16: for $i = 1$ to $n$ do
17:   for $j = i$ to $n$ where $G'(i, j) \neq 0$ do
18:     $U'_{kk}(i) = G'(i, i)$;
19:     $U'(i, j) = G'(i, j)/G'(i, i)$;
20:   end for
21: end for
```

3) SUBTASK SCHEDULING FOR THE SUBSTITUTIONS

Generally, executing the subtasks of the substitutions in order will attain the desired solution $\boldsymbol{u}'$. However, it is time-consuming and needs to be improved for real-time application. Thus the parallelism existing in both the forward and backward substitutions should be found. Fortunately, some subtasks are independent due to the zeros of $\boldsymbol{L}'$ and $\boldsymbol{U}'$. To maximize the fine-grained parallelism potential of the substitutions, a strict scheduling of the subtasks is presented.

Firstly, we note that the time consuming of each floating-point operation T_{op} is fixed. For non-zero element L_{ij}, the start time of the corresponding floating-point operation is T_{ij}. For column i, its start time is T_i. Then all the fill-ins of this column are injected into the PEs as a pipeline. One subtask is accomplished by one column operation. It should be noted that all the operations are driven by the physical clock of FPGA. For the first column of $\boldsymbol{L}'$, it is stated that the participation operation time of the first non-zero is at the first clock period and obviously the participation operation time of the nth non-zero element is at nth clock period due to the pipelined dataflow.

Taking the scheduling of the forward substitution as an example, the main steps are as follows:

1) Analyze the dependencies of the non-zeros and determine the operation start time of each non-zero. For the non-zero L_{ij} and L_{mn}, we state that they are interdependent if it meets the following condition:

$$\{i = m \,||\, i = n \,||\, j = m \,||\, j = n\}$$

1) Determine the initial start time of each column operation based on the start time of the non-zeros in this column;
2) All the column operations are assigned to the PEs in a time sequence based on Round-Robin. The column operation is delayed when it comes and all the PEs are busy. And the start time of the dependent column operations to be unassigned delays the same clock periods;
3) Store the updated start time of each column operation and the element processing number corresponding to the column operation.

To illustrate how the topology analysis works, we take the matrix $\boldsymbol{L}$ shown in Fig. 2 as an example. The number in the grid denotes the start time of the corresponding non-zero and the shaded arrow denotes the dependencies between non-zeros. For the non-zero L_{ij}, the formula $i_j^i = i_j^{i-1} - L_{ij}y_i$ need to be performed. As on FPGA the solving procedure includes one multiplication, one addition and one assignment, and each floating-point operation consumes 5, 7, 1 clock periods respectively, the entire procedure consumes 13 clock periods. Obviously, the start time of the formula for L_{ij} relies on the finishing time of i_j^{i-1} and y_i. The columns required at any step of the analysis process are represented by the shaded portion of the matrix and the process is accomplished from the left column to the right column.

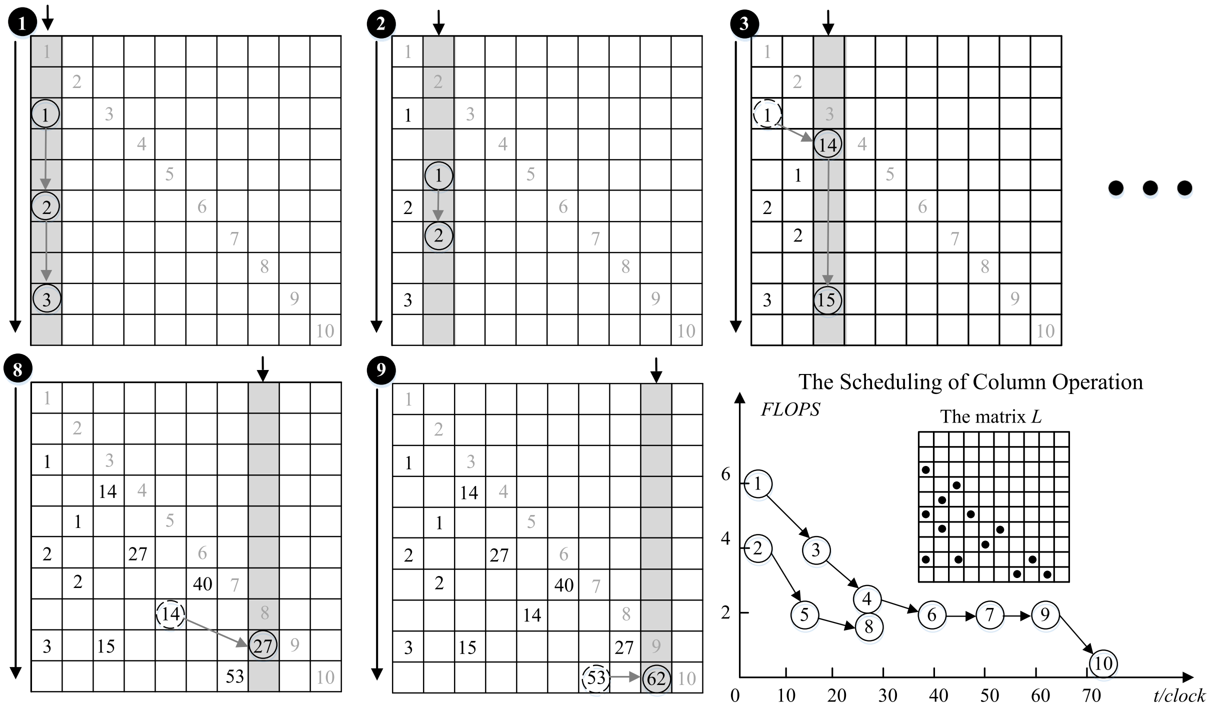


FIGURE 2. Topology analysis of the example matrix L.

Firstly, we give the start time of the non-zeros in column 1: $T_{31} = 1, T_{61} = 2, T_{91} = 3$. For the non-zeros in column 2, they are independent with the non-zeros in column 1, hence $T_{52} = 1, T_{91} = 2$. For the non-zeros in column 3, the start time of L_{43} relies on T_{31}, hence $T_{43} = T_{31} + 13 = 14$. Similarly, the start time of L_{93} relies on T_{43} and T_{91}, hence $T_{93} = max\{T_{43}+1, T_{91}+9\} = 15$. Then the start time pattern of the non-zeros in matrix $\boldsymbol{L}$ can be calculate as Fig. 2 shows.

Next the initial start time of the column operation T_i for column i is determined by (11).

$$T_i = max\{T_{ij} - m\} \tag{11}$$

where m denotes the offset of non-zero L_{ij} from the first non-zero in column i. For example, $m = 0$ for L_{31} and $m = 1$ for L_{61}. For the column 1, $T_1 = \max\{T_{31} - 0, T_{61} - 1, T_{91} - 2\} = 1$. After all the initial start time of the column operations are calculated, the scheduling of each column operation $T = \{1, 1, 14, 27, 14, 40, 53, 27, 62, 71\}$ is obtained. Meanwhile, the column operations are divided into two groups according to the dependencies shown in Fig. 2, where the abscissa denotes the start time and the ordinate denotes the number of the floating-point operations for each column. In detail, $Col_{g1} = \{1, 3, 4, 6, 7, 9, 10$ and $Col_{g2} = \{2, 5, 8\}$, where the figures denote the column number.

Then the column operations are assigned to the PEs. It is noted that the start time of the column operations changes with the total number of PEs employed in the solver. Explanatorily, P_i denotes the number of the PE that implements the column operation for column i. If only one PE is used, $P = \{1, 1, 1, 1, 1, 1, 1, 1, 1, 1\}$. Thus the column operations for column 2 cannot be assigned into the PE until the operation for column 1 is finished. Hence, $T_2 = 4$ and the operations depending on column 2 are delayed the same time to avoid data conflicts. It is the same with the other columns. Consequently, the scheduling of each column operation is updated as $T = \{1, 4, 14, 27, 17, 40, 53, 30, 62, 71\}$. When more than one PE is used in the solver, two groups of columns can be processed in parallel, hence $T = \{1, 1, 14, 27, 14, 40, 53, 27, 62, 71\}$, $P = \{1, 2, 1, 1, 2, 1, 1, 2, 1, 1\}$. It improves the flexibility and parallelism for the scheduling of column operations.

Finally, T and P are pre-stored in the shared memory as initial data that will be downloaded into FPGA.

B. ONLINE DESIGN

The online design on FPGA is to realize the forward and backward substitutions. In this section, the compressed sparse formats for the upper triangular matrix $\boldsymbol{U}$ and the lower triangular matrix $\boldsymbol{L}$ are firstly stressed. Then the detailed splitting of the column operations for the substitutions implemented on FPGA is presented.

1) COMPRESSED SPARSE FORMAT OF L' AND U'

The matrix $\boldsymbol{L}'$ and $\boldsymbol{U}'$ are generally sparse matrices in ADNs [27]. To reduce the hardware storage resources, the compressed sparse formats are used for the storage of $\boldsymbol{L}'$ and $\boldsymbol{U}'$. Compressed sparse row (CSR) and compressed sparse column (CSC) are the most common formats for sparse matrices [24]. As the forward and backward substitution are implemented by column operations, the matrix $\boldsymbol{L}'$ and $\boldsymbol{U}'$ are stored using CSC format to simplify the design. The CSC format consists of three arrays as follows:

1) *val* array. It is a floating-point array which stores the value of the non-zeros by column;
2) *row_ind* array. It is an integer array which stores the row index for the non-zeros in *val* array;
3) *col_ptr* array. It is an integer array. *col_ptr*[*i*] denotes the offset of the first non-zero in column *i*. *len*[*i*] denotes the count of the non-zeros in column *i*, which can be calculated by (12).

$$len\,[i] = col\_ptr\,[i+1] - col\_ptr\,[i] \tag{12}$$

The above three arrays are stored in the shared memory as the offline design states. Specially, *val* array and *row_ind* array are stored as a 72-bit data frame to facilitate parallel processing as shown in Fig. 3.

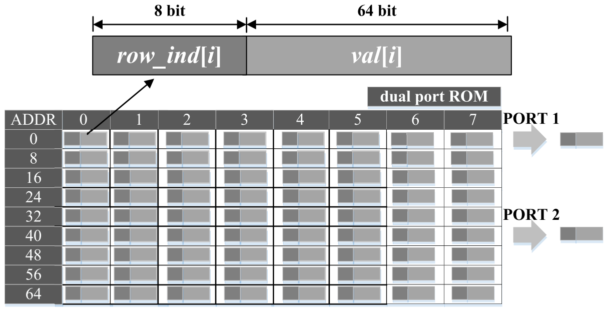


FIGURE 3. Compressed matrix storage example.

2) COLUMN OPERATION SPLITTING

In this paper, the online forward and backward substitutions are accomplished by (9) and (10). The substitution algorithm is described in Algorithm 2.

C. HARDWARE IMPLEMENTATION

Hardware design of the kernel solver for the FPGA-based RT simulator is shown in Fig. 4. The simulator consists of global control module, electrical system solution module, control system solution module, data interaction module and I/O interface [28]. The solver is embedded in the electrical system solution module.

The column operations for the forward and backward substitutions are assigned to the corresponding PEs. Each PE has the universal structure that is constructed by a multiplier, a subtractor, a delayer and a start timer. The number of PEs can be configured flexibly in accordance with the scale of $\boldsymbol{L}'$ and $\boldsymbol{U}'$. The output data of the PE are utilized to update the vector $\boldsymbol{y}'$ and $\boldsymbol{u}'$ to be solved. All the initial data are stored in shared read only memory (ROM). *Global solve*

Algorithm 2 Forward and Backward Substitutions Processes

```
1:  /* Forward substitution *
2:  for $j = 1$ to $n$ do
3:      $y'(j) = i'(j)$;
4:      for $k = j+1$ to $n$ where $L(k,j) \neq 0$ do
5:          $i'(k) = i'(k)$-$L(k,j) * y'(j)$;
6:      end for
7:      $y'(j) = y'(j)/U_{kk}(j)$;
8:  end for
9:  /* Backward substitution */
10: for $j = n$ to 1 do
11:     $u'(j) = y'(j)$;
12:     for $k = j$-1 to 1 where $U'(k,j) \neq 0$ do
13:         $y'(k) = y'(k)$-$U'(k,j) * u'(j)$;
14:     end for
15: end for
```

control is utilized to coordinate the operation of dataflow bus, PE and ROM. *Data selector* is utilized to select the corresponding PE for each column operation according the pre-stored data.

To illustrate how the online process works, the forward substitution assigned to the PE is illustrated as shown in Fig. 4. Firstly, the *start timer* and *Data selector* are initialized by the pre-stored data of the scheduling datasheet. Meanwhile, the vector of $\boldsymbol{i}$ is reordered according to the addresses generated by COLAMD algorithm. Then the start signal *sta* is set to one, and *start timer* begins to count. When a certain timer counts to $'1'$, the corresponding enable signal *control_y_ena*[K] for column operation K is set to one. Simultaneously, the non-zeros $L[K]$ and their row indexes *L_row*[K] are injected into the PE in parallel to finish the floating-point operations with the reordered $\boldsymbol{i}'$. The results are temporarily stored in registers *i_temp*[K] and subsequently used to update $\boldsymbol{i}'$ by the bus *i & L memory bus*. In addition,*Delay_nclk* module is utilized to align data and control signal.

The backward substitution process is similar with the forward substitution process. Whereas, the vector of $\boldsymbol{U}$ needs to be reordered before involved into the operation of the other modules in the simulator.

IV. CASE STUDY

To validate the solver, a modified IEEE 33-node system with PVs is implemented on the 4-FPGA-based real-time simulator. Fig. 5 shows the simulation platform and Fig. 6 shows the test case. The performance of the solver is investigated by the minimum time-step with respect to the simulated system size. The simulation results are compared with PSCAD/EMTDC® to verify the accuracy of the solver. To further highlight the performance of the LU decomposition based solver, the inverse matrix pre-calculation method has also be realized and executed on the FPGA. The contrastive analysis are conducted in detail.

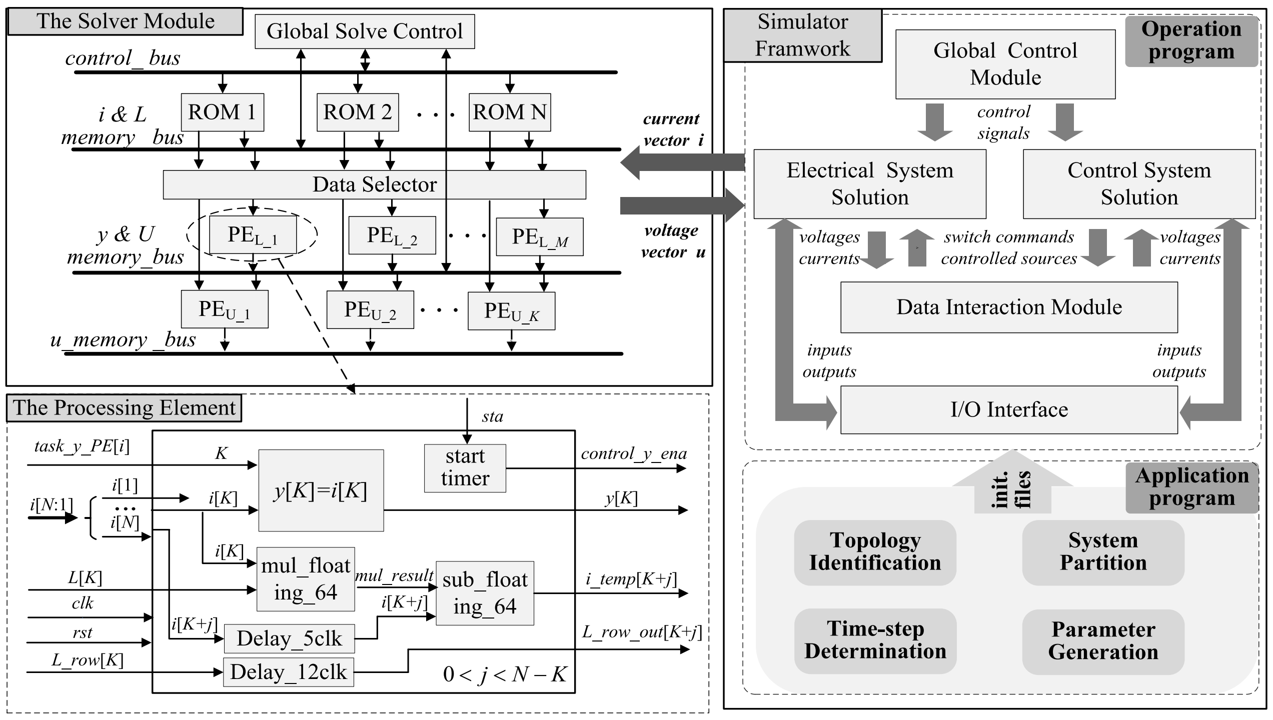


FIGURE 4. Hardware design of the kernel solver of FPGA-based real-time simulator.

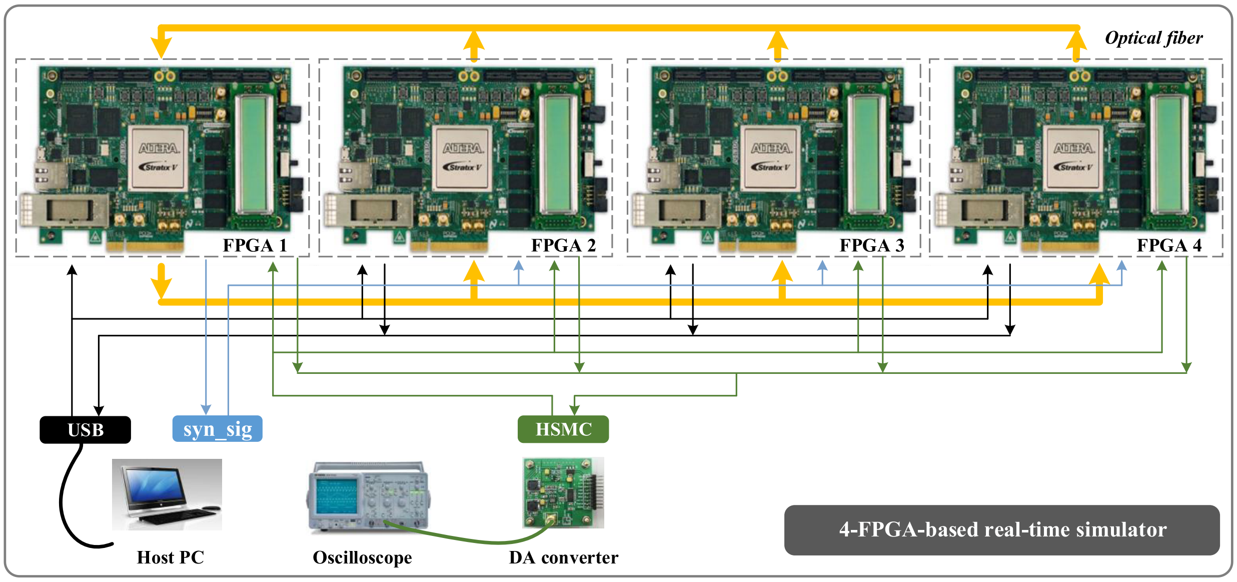


FIGURE 5. The 4-FPGA-based real-time simulator.

A. SIMULATION PLATFORM

Fig. 5 provides the overview of the FPGA-based real-time simulator. Four Stratix® V Edition DSP development boards from Altera® are utilized. They are connected directly by optical fibers through Quad Small Form-factor Pluggable (QSFP). FPGA 1 is treated as the master FPGA, which provides synchronization signals for the other three FPGAs. The entire design on FPGAs are driven by a 125MHz clock which determines the execution time of the design.

B. TEST CASE

Fig. 6 shows the structure of the modified IEEE 33-node system with PVs, which consists of a network with 33 three-phase nodes and three PV units, connecting to node 18, node 22, node 33 respectively. The detailed structure of the PV unit is shown in Fig. 7. Line types and the respective parameters are given in [29]. The PV array is modeled by a single diode equivalent circuit [30]. The PV array is directly connected to the DC bus. The inverter is controlled by the

TABLE 1. Numbers of different types of elements for the subsystems.

Subsystem	Supply source	Controlled source	RLC	IGBT	Diode	Meter	Line	Transformer
network (FPGA 1)	3	0	96	0	0	0	96	0
PV 1 (FPGA 2)	0	2	8	6	6	8	0	3
PV 2 (FPGA 3)	0	2	8	6	6	8	0	3
PV 3 (FPGA 4)	0	2	8	6	6	8	0	3

TABLE 2. The scheduling datasheet for PV unit.

Operation	Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Forward substitution	Start time	1	14	27	1	14	27	1	14	27	40	53	66	79	92	105
	FLOPs	2	2	2	2	2	2	2	2	2	4	6	6	4	4	0
Backward substitution	Start time	67	54	41	39	38	26	92	91	79	66	53	40	27	14	1
	FLOPs	0	2	2	0	4	0	2	2	0	4	0	2	2	0	4

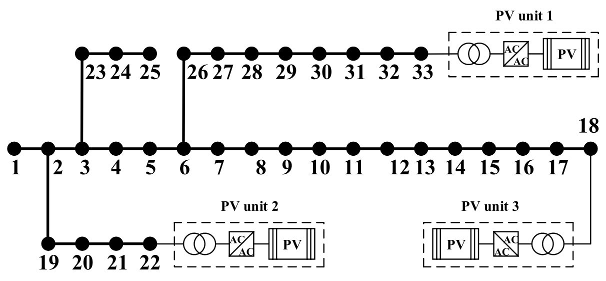


FIGURE 6. The IEEE 33-node system with PVs.

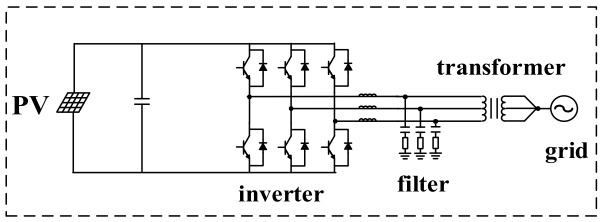


FIGURE 7. The detailed structure of the PV unit.

V_{dc}-Q strategy, in which the referent DC voltage is provided by the maximum power point tracking (MPPT) algorithm [31]. The power factor is set to 1.0.

The modified IEEE 33-node system with PVs is portioned into four subsystems, including a 33-node system and three PV units. Four subsystems are simulated on four interconnected FPGAs. The entire system contains 3 supply source elements, 6 controlled source elements, 120 RLC elements, 18 IGBTs, 18 diodes, 24 meters, 96 lines and 9 transformers. The numbers of different types of the elements in the four subsystems are shown in Table 1.

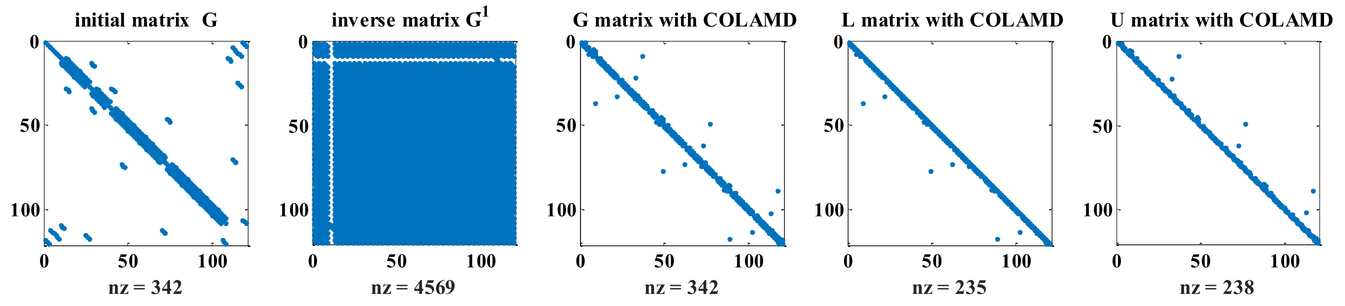
TABLE 3. The scheduling of the PEs for PV unit.

process	PE number	Column operation number	Time consumption/clock
Forward substitution	1	{1-6}	32
	2	{7-15}	107
Backward substitution	1	{1-6}	69
	2	{7-15}	94

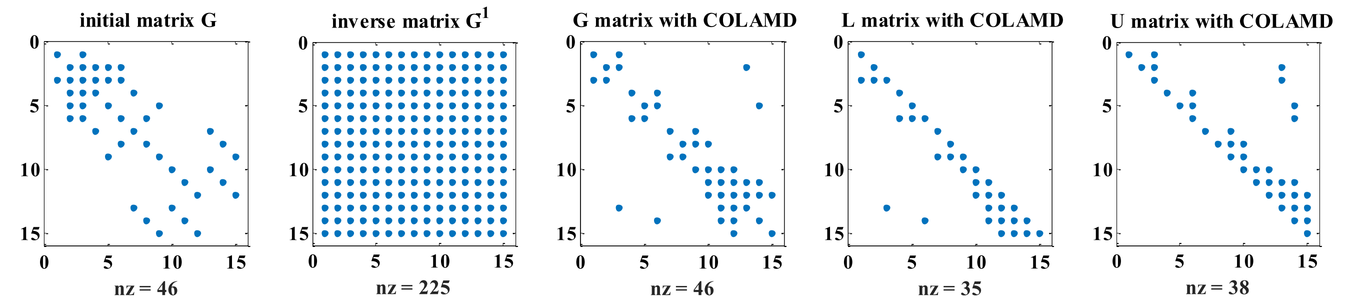
C. CASE ANALYSIS

In the modified IEEE 33-node system with PVs, the matrix dimension of the network is 120 and the matrix dimension of each PV unit is 15. The results of the COLAMD algorithm are shown in Fig. 8. For the 33-node system and PV unit, the original conductance matrices cannot be LU decomposed directly. After LU decomposition with COLAMD algorithm, the matrices can be decomposed. For the PV unit, the count of non-zeros in matrix $\boldsymbol{L}'$ and $\boldsymbol{U}'$ are 35 and 38 respectively. For the network, the count of non-zeros in matrix $\boldsymbol{L}'$ and $\boldsymbol{U}'$ are 235 and 238 respectively. The count of non-zeros in the inverse matrix $\boldsymbol{G}^{-1}$for the 33-node system and the PV unit are 4569 and 225 respectively. It is noted that the inverse matrices are almost full rank.

The scheduling of subtasks gives the initial start time of each column operation for the subsystems. Table 2 shows the scheduling datasheet of the PV unit in detail. To balance the time and resources consumption, the optimal scheduling of the PEs for PV unit and 33-node system are shown in Table 3 and Table 4 respectively. For the PV unit, 4 PEs are consumed, and for the network, 9 PEs are consumed. All the PEs are implemented in parallel, which takes full advantage of the dataflow processing pattern on FPGA.



(a) The IEEE 33-node network



(b) The PV unit

FIGURE 8. Results of the COLAMD algorithm (a) The IEEE 33-node network. (b) The PV unit.

TABLE 4. The scheduling of the PEs for the 33-node system.

process	PE number	Column operation number	Time consumption / clock
Forward substitution	1	{1-22}	170
	2	{23-40}	263
	3	{41-62}	170
	4	{63-80}	263
	5	{81-102}	170
	6	{103-120}	250
Backward substitution	1	{1-40}	252
	2	{41-80}	258
	3	{81-120}	247

TABLE 5. Resource utilization for the subsystems.

Subsystem		Logic resources	DSP blocks	Memory bits
Network	Solver	67%	4%	0.7%
	Others	21%	0%	7.9%
PV 1	Solver	14%	2%	0.3%
	Others	5%	7%	2.2%
PV 2	Solver	14%	2%	0.3%
	Others	5%	7%	2.2%
PV 3	Solver	14%	2%	0.3%
	Others	5%	7%	2.2%

TABLE 6. Execution time of each subsystem within one time-step.

Subsystem	Solver module			Other modules
	Forward substitution	Backward substitution	Others	
Network	2.104 µs	2.064 µs	1.960 µs	4.384µs
PV 1	0.856 µs	0.752 µs	0.280 µs	1.688 µs
PV 2	0.856 µs	0.752 µs	0.280 µs	1.688 µs
PV 3	0.856 µs	0.752 µs	0.280 µs	1.688 µs

Table 5 lists the main hardware resource utilization of the four subsystems. As can be seen, the logic resources have been mostly utilized for the 33-node system and the hardware of the FPGA 2, FPGA 3 and FPGA 4 for PVs still have enough room for further expansion. Compared with other modules, resources consumed by the solver are perfectly acceptable.

The execution time of each subsystem within a simulation time-step is shown in Table 6. The total time consumption of each PV and the 33-node system is $3.576\mu s$ and $10.512\mu s$ respectively. The time consumption of the solver is acceptable for real-time simulation. The selected time-steps for PV and the 33-node system are $4\mu s$ and $12\mu s$ respectively.

The inverse matrix pre-calculation is one of the current real-time linear equations solving methods [32], [33]. The nodal equation is solved by multiplying the pre-calculated

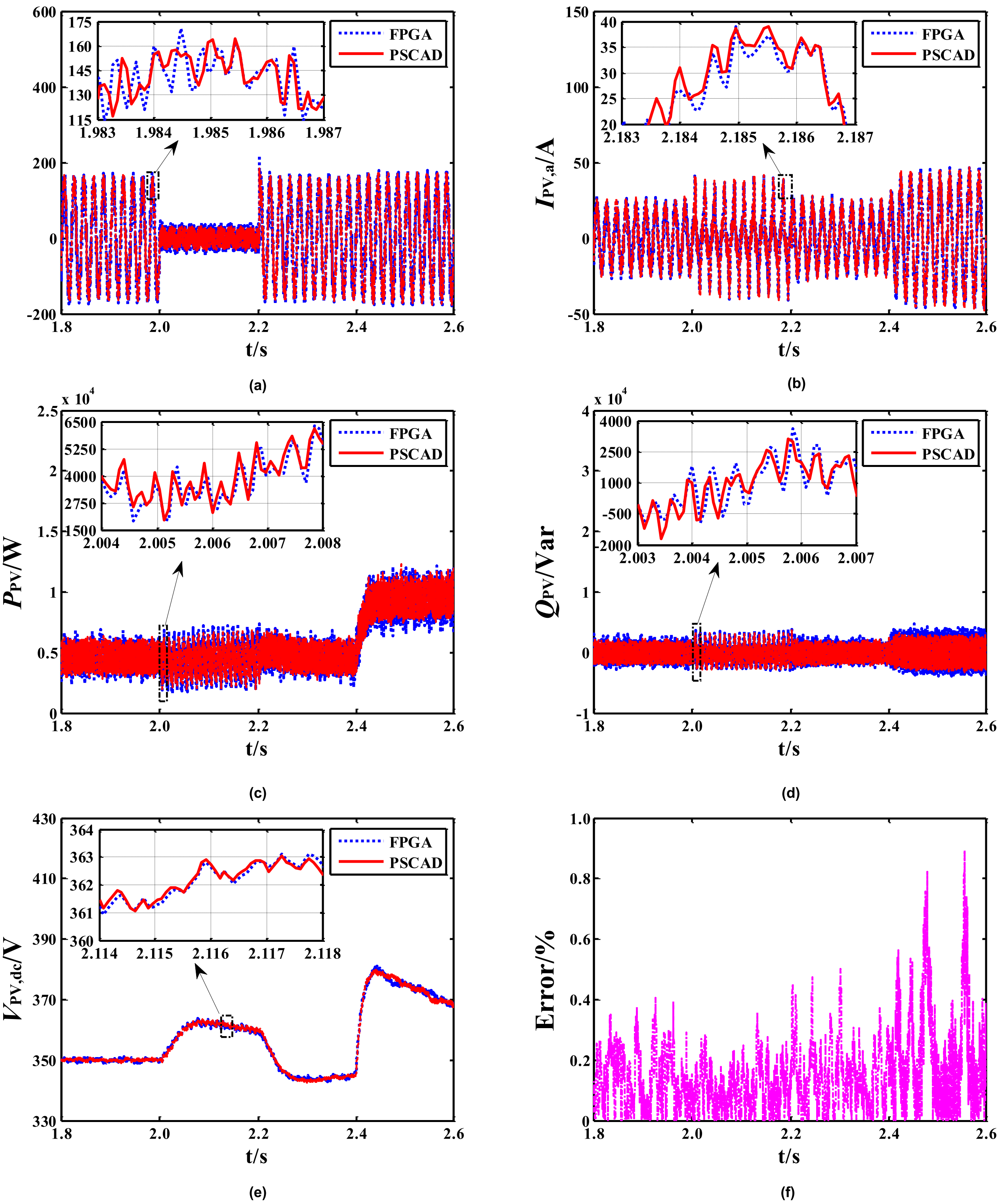


FIGURE 9. Simulation results of test case. (a) Phase-A voltage waveform of PV unit 2 (b) Phase-A current waveform of PV unit 2. Active power output waveform of PV unit 2 (d) Reactive power output waveform of PV unit 2. (e) DC voltage waveform of PV unit 2 (f) Relative error of the DC voltage.

inverse nodal conductance matrix by the node injected currents. All the inverse matrices, corresponding to the possible network topologies resulting from switching operations, are calculated and stored before the real-time simulation process. The hardware resource and execution time consumption of the network solver based on the inverse matrix pre-calculation as well as the LU decomposition, for the 33-node system, are given in Table 7.

As can be seen from Table 7, the time consumption of the LU decomposition based solver is larger than the inverse matrix pre-calculation based one, since the number formats of the two solvers are different. The addition operations in the first solver are performed using the floating-point format, while the second solver uses fix-point format. The floating-point addition operation is more time-consuming but more precise so as to reduce the rounding error of the direct method. It should be noted that the execution time of the two solvers are on the same order and are both appropriate for the RT simulation. In addition, the hardware resource consumption of the former is much smaller than the latter, which demonstrates the superiority of the LU decomposition based solver.

TABLE 7. Comparison of the resources used with different kernel solvers for the 33-node system.

Solver	Hardware resource consumption			Time consumption
	Logic resources	DSP blocks	Memory bits	
LU decomposition	67%	4%	0.7%	6.128 µs
Inverse matrix pre-calculation	79%	31%	16.7%	1.200 µs

D. SIMULATION RESULTS

In the test case, two events are as follows to reproduce the transients: i) At 2.0 second, a Phase-A ground fault occurs at node 22 and lasts 0.2 second. ii) At 2.4 second, the irradiance arises from 500W/m^2 to 1000W/m^2. The simulation results are compared with PSCAD/EMTDC® to validate the performance of the real-time solver. The time-step of PSCAD/EMTDC is 4.0μs. The waveforms of the test case are shown in Fig. 9(a)-(f). Fig. 9(a) and Fig. 9(b) plot the Phase-A voltage and Phase-A current waveforms of PV unit 2. During the fault, the Phase-A voltage drops from 150V peak to 0V at 2.0 second and recovers back at 2.2 second. The Phase-A current raises with the irradiance increasing at 2.4 second. Fig. 9(c)-(f) show the active power output, reactive power output, DC voltage and its relative error of PV unit 2.

As can be seen from Fig. 9, the results of the two simulation tools are nearly the same, which verifies the accuracy and the correctness of real-time solver design. The errors of the FPGA-based simulator are introduced by the following aspects:

i) The modelling method for power electronic devices are different in FPGA and PSCAD®. In FPGA, the device is represented by the associated discrete circuit (ADC) [34], while in PSCAD/EMTDC®, it is represented by large-small resistance;

ii) In FPGA, the entire system is decoupled due to the hardware resources limitation. In PSCAD/EMTDC® it is solved as a whole;

iii) Different time-steps are used in the FPGA-based simulator while the same time-step is used in PSCAD/EMTDC®.

V. CONCLUSIONS

This paper proposed a high-performance solver design for the FPGA-based real-time simulator, which is a key computational kernel of the matrix solution phase. A fine-grained framework of the solver is designed for the multi-FPGA system, which enables full use of hardware resources. An offline design on PC and an online design on FPGA are presented in detail, which maximizes the parallelism potential of the solver. The IEEE 33-node system with PVs is simulated to validate the solver in the simulator. The performance of the solver in the offline process and online process is analyzed in detail, and both processes design are further verified. The results are compared with PSCAD/EMTDC under the same conditions, which verifies the correctness of the solver.

REFERENCES

[1] M. D. O. Faruque *et al.*, "Real-time simulation technologies for power systems design, testing, and analysis," *IEEE Power Energy Technol. Syst. J.*, vol. 2, no. 2, pp. 63–73, Jun. 2015.

[2] M. Hernandez *et al.*, "Embedded real-time simulation platform for power distribution systems," *IEEE Access*, vol. 6, pp. 6243–6256, Dec. 2017.

[3] S. Vavilapalli *et al.*, "Design and real-time simulation of an AC voltage regulator based battery charger for large-scale PV-grid energy storage systems," *IEEE Access*, vol. 5, pp. 25158–25170, Oct. 2017.

[4] X. Guillaud *et al.*, "Applications of real-time simulation technologies in power and energy systems," *IEEE Power Energy Technol. Syst. J.*, vol. 2, no. 3, pp. 103–115, Sep. 2015.

[5] J. Mahseredjian, V. Dinavahi, and J. A. Martinez, "Simulation tools for electromagnetic transients in power systems: Overview and challenges," *IEEE Trans. Power Del.*, vol. 24, no. 3, pp. 1657–1669, Jul. 2009.

[6] C. Wang, X. Fu, P. Li, J. Wu, and L. Wang, "Multiscale simulation of power system transients based on the matrix exponential function," *IEEE Trans. Power Syst.*, vol. 32, no. 3, pp. 1913–1926, May 2017.

[7] Y. Chen and V. Dinavahi, "FPGA-based real-time EMTP," *IEEE Trans. Power Del.*, vol. 24, no. 2, pp. 892–902, Apr. 2009.

[8] X. Wang, S. Mou, and D. Sun, "Improvement of a distributed algorithm for solving linear equations," *IEEE Trans. Ind. Electron.*, vol. 64, no. 4, pp. 3113–3117, Apr. 2017.

[9] P. Wang, W. Ren, and Z. Duan, "Distributed algorithm to solve a system of linear equations with unique or multiple solutions from arbitrary initializations," *IEEE Trans. Control Netw. Syst.*, to be published.

[10] T. Han and Y. Han, "Numerical solution for super large scale systems," *IEEE Access*, vol. 1, pp. 537–544, Aug. 2013.

[11] J. Liu, Y. Liang, and N. Ansari, "Spark-based large-scale matrix inversion for big data processing," *IEEE Access*, vol. 4, pp. 2166–2176, 2016.

[12] W. L. Whipple, "Linear equation solving for continuous simulation," *ACM SIGSIM Simul. Dig.*, vol. 3, no. 2, pp. 12–16, Jan. 1972.

[13] C. Fu, J. D. McCalley, and J. Tong, "A Numerical solver design for extended-term time-domain simulation," *IEEE Trans. Power Syst.*, vol. 28, no. 4, pp. 4926–4935, Nov. 2013.

[14] A. Amritkar, E. de Sturler, K. Swirydowicz, D. Tafti, and D. Tafti, "Recycling Krylov subspaces for CFD applications and a new hybrid recycling solver," *J. Comput. Phys.*, vol. 303, pp. 222–237, Sep. 2015.

[15] A. Ebrahimi and M. Zandsalimy, "Evaluation of FPGA hardware as a new approach for accelerating the numerical solution of CFD problems," *IEEE Access*, vol. 5, pp. 9717–9727, May 2017.

[16] Q. Chen, X. Luo, L. Zhang, and S. Quan, "Model predictive control for three-phase four-leg grid-tied inverters," *IEEE Access*, vol. 5, pp. 2834–2841, Feb. 2017.

[17] G. Wu, Y. Dou, J. Sun, and G. D. Peterson, "A high performance and memory efficient LU decomposer on FPGAs," *IEEE Trans. Comput.*, vol. 61, no. 3, pp. 366–378, Mar. 2012.

[18] M. K. Jaiswal and N. Chandrachoodan, "FPGA-based high-performance and scalable block LU decomposition architecture," *IEEE Trans. Comput.*, vol. 61, no. 1, pp. 60–72, Jan. 2012.

[19] Y. Chen and V. Dinavahi, "An iterative real-time nonlinear electromagnetic transient solver on FPGA," in *Proc. IEEE PESGM*, Detroit, MI, USA, Jul. 2011, pp. 1–8.

[20] R. H. Bartels and G. H. Golub, "The simplex method of linear programming using LU decomposition," *ACM Commun.*, vol. 12, no. 5, pp. 266–268, May 1969.

[21] E. Angerson *et al.*, "LAPACK: A portable linear algebra library for high-performance computers," in *Proc. ACM ICS*, New York, NY, USA, Nov. 1990, pp. 2–11.

[22] T. Davies, C. Karlsson, H. Liu, C. Ding, and Z. Chen "High performance linpack benchmark: A fault tolerant implementation without checkpointing," in *Proc. ACM ICS*, Tucson, AZ, USA, 2011, pp. 162–171.

[23] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, "Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm," *ACM Trans. Math. Softw.*, vol. 30, no. 3, pp. 377–380, Sep. 2004.

[24] P. T. Stathis, "Sparse matrix vector processing formats," Ph.D. dissertation, Dept. Comput. Eng., Delft Univ. Technol., Delft, The Netherlands, 2004.

[25] J. Mahseredjian, "On a new approach for the simulation of transients in power systems," in *Proc. ICPST*, Montreal, QC, Canada, 2005, pp. 1–6.

[26] T. A. Davis and E. P. Natarajan, "Algorithm 907: KLU, a direct sparse solver for circuit simulation problems," *ACM Trans. Math. Softw.*, vol. 37, no. 3, pp. 1–17, Sep. 2010.

[27] N. Watson and J. Arrillaga, *Power Systems Electromagnetic Transients Simulation*. London, U.K.: IEE, 2003.

[28] P. Li, Z. Wang, C. Wang, X. Fu, H. Yu, and L. Wang, "Synchronisation mechanism and interfaces design of multi-FPGA-based real-time simulator for microgrids," *IET Gener., Transmiss. Distrib.*, vol. 11, no. 12, pp. 3088–3096, Aug. 2017.

[29] M. E. Baran and F. F. Wu, "Optimal capacitor placement on radial distribution systems," *IEEE Trans. Power Del.*, vol. 4, no. 1, pp. 725–734, Jan. 1989.

[30] P. Li, Z. Wang, C. Ding, H. Yu, and C. Wang, "A design of grid-connected PV system for real-time transient simulation based on FPGA," in *Proc. IEEE PESGM*, Denver, CO, USA, Jul. 2015, pp. 1–5.

[31] T. Esram and P. L. Chapman, "Comparison of photovoltaic array maximum power point tracking techniques," *IEEE Trans. Energy Convers.*, vol. 22, no. 2, pp. 439–499, Jun. 2007.

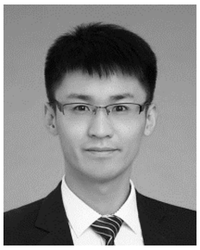
[32] *Power Time Digital Simulator Power System User's Manual*, RTDS Technol. Inc., Winnipeg, MB, Canada, 2012.

[33] C. Dufour, H. Saad, J. Mahseredjian, and J. Bélanger, "Custom-coded models in the state space nodal solver of ARTEMiS," in *Proc. IPST*, Vancouver, BC, Canada, 2013, pp. 1–6.

[34] T. Maguire and J. Giesbrecht, "Small time-step ($< 2\mu$sec) VSC model for the real-time digital simulator," in *Proc. IPST*, Montreal, QC, Canada, 2005, pp. 1–6.

ZHIYING WANG was born in Shuozhou, Shanxi, China, in 1990. She received the B.S. degree in electrical engineering from Tianjin University, Tianjin, China, in 2013.

She is currently pursuing the Ph.D. degree with the School of Electrical and Information Engineering, Tianjin University. Her current research interests include the real-time simulation and analysis of distributed generation systems, microgrids, smart grid, and active distribution networks.



FANPENG ZENG was born in Yingkou, Liaoning, China, in 1992. He received the B.S. and M.S. degrees in electrical engineering from Tianjin University, Tianjin, China, in 2015 and 2018, respectively.

He is currently with State Grid Tianjin Dongli Electric Power Supply Company, Tianjin. His current research interests include the real-time simulation of distributed generation systems, smart grid, and active distribution networks

PENG LI was born in Tianjin, China, in 1977. He received the B.S. and Ph.D. degrees in electrical engineering from Tianjin University, Tianjin, in 2004 and 2010, respectively.

He is currently an Associate Professor with the School of Electrical and Information Engineering, Tianjin University. His current research interests include the simulation and analysis of distributed generation systems, microgrids, smart distribution systems, and active distribution networks.

CHENGSHAN WANG (SM'11) was born in Tianjin, China, in 1962. He received the Ph.D. degree in electrical engineering from Tianjin University, Tianjin, in 1991.

From 1994 to 1996, he was a Senior Academic Visitor with Cornell University, Ithaca, NY, USA. From 2001 to 2002, he was a Visiting Professor with Carnegie Mellon University, Pittsburgh, PA, USA. He is currently a Professor with the School of Electrical and Information Engineering, Tianjin University, where he is also the Director of the Key Laboratory of Smart Grid of the Ministry of Education. His current research interests include distribution system analysis and planning, distributed generation systems and microgrids, and power system security analysis.

XIAOPENG FU (M'17) received the B.S. and Ph.D. degrees in electrical engineering from Tianjin University, Tianjin, China, in 2011 and 2016, respectively.

He is currently a Lecturer with the School of Electrical Engineering and Automation, Tianjin University. His current research interests include the analysis of distributed generation systems, microgrids, and electromagnetic transient's simulation.

JIANZHONG WU (M'14) received the Ph.D. degree from Tianjin University, Tianjin, China, in 2004.

From 2004 to 2006, he was with Tianjin University, where he was an Associate Professor. From 2006 to 2008, he was a Research Fellow with The University of Manchester, Manchester, U.K. He is currently a Professor with the School of Engineering, Institute of Energy, Cardiff University, Cardiff, U.K. His current research interests include energy infrastructure and smart grids.