Journal of Control, Automation and Electrical Systems (2022) 33:677–687
https://doi.org/10.1007/s40313-021-00842-6

Linear Equation System Solver Circuit for Streaming Processing in FPGA

Tiago A. Teixeira[1] · **Luciano M. de Andrade Filho**[1]

Received: 16 April 2021 / Revised: 7 August 2021 / Accepted: 6 October 2021 / Published online: 26 October 2021

Abstract
The solution of linear equation system (LES) has numerous applications in digital signal processing. However, due to its high computational cost, it becomes difficult to implement it in high-speed free-running (streaming) processing applications. This work presents an implementation, in FPGA, of a digital circuit for LES resolution, using the gradient descent method, capable of operating online. Such implementation brings a new perspective of execution of iterative algorithms, based on matrix calculations, in embedded processing for high-speed (tens of MHz) uninterrupted data-flow. In other words, the proposed method acts as a real-time processing block (at each clock, a new data enters and another one leaves), allowing the implementation of matrix calculations in a transparent way for continuous flow systems. In addition to proposing an implementation architecture, this work discusses the relationship between the number of iterations, the amount of resources used and the allowable acquisition rate. As a case study, the method is applied to a channel equalization problem.

Keywords Linear equation system · FPGA · Online processing · Channel equalization

1 Introduction

The need to solve linear equation systems (LES) has been finding allies among the computer technologies developed in recent years. One of these allies, within the area of digital signal processing, has been the development of specific digital circuits to solve LES.

Many proposals have been made using different computing approaches. Examples based on distributed network can be found in Fedasyuk et al. (2009) and Yu et al. (2016). Other approaches use a single computer, where the solvers are built over CPU-GPU platform (CPU—Central Processor Unit; GPU—Graphic Processor Unit), like the work in Binotto et al. (2010). The work in Khun et al. (2015) introduced a LES solver optimized for a GPU. A comparison between solvers using GPU and FPGA (Field Programmable Gate Array) (Kuon et al. 2008) is made in Salah and AbdelSalam (2017). Examples of a LES resolution approach, that uses FPGA, can be found in (Sudarsanam and Aravind 2005; Yang et al. 2009; Martinez-Alonso et al. 2010)

All of these cited works have a common characteristic: they work with batched data structure as input–output data-flow. This kind of structure, where it is necessary to load a big amount of data, reduces the overall data processing ratio (for some kHz) in applications where continuous data-flow is necessary. In this work, it will be presented a way to use FPGA to solve LES using uninterrupted high-rate data obtained direct from real-time digital systems, like free-running Analog-to-Digital Converters (ADC), allowing the use of LES solver in real-time Digital Signal Processing (DSP) (Mitra 2010) applications.

In (DSP), iterative LES resolution methods are widely used (Xu et al. 1994; Luenberger and Ye 2008; Saad 2003; Kelley 1995). Among these methods, the most cost-effective for embedded FPGA systems is based on Gradient Descent (GD) (Luenberger and Ye 2015) algorithms, because it requires adders and multipliers circuitry only. Even so, the need for matrix calculations and a large number of iterations discourages its usage in high-speed Digital Continuous Flow Systems (DCFS) (Akidau et al. 2018).

DCFSs operate on uninterrupted discrete signals. The implementation of LES resolution-based algorithms is a big challenge and also it is of great interest to the DCFS. In many

✉ Tiago A. Teixeira
tiago.teixeira@engenharia.ufjf.br

Luciano M. de Andrade Filho
lucianom@cern.ch

[1] Universidade Federal de Juiz de Fora, Juiz de Fora, Brazil

of these free-running systems, for example, in digital communication (Proakis 2007), the information is grouped in a sequence of temporal samples, of fixed size, for the transmission of symbols that carry the encoded information. Digital systems with fixed size windows, such as the one mentioned above, can greatly benefit from LES resolution processing that could be transparent to the free-running characteristic of the input data. Among some examples, we can mention the use of Transforms such as Fourier (Oppenheim 2010) and Wavelets (Diniz 2010) and their inverse transformations, Principal Component Analysis (Jolliffe 2002) and Whitening (Trees 2013), Orthogonalization and Diagonalization of Matrices (Golub and Van Loan 1996), Adaptive Filters based on Recursive Least Square (Diniz 2006), Channel Equalization in digital communication (Gerstacker et al. 2004), etc.

This paper presents a LES solution implementation that can be used in high-speed free-running systems (tens of MHz) in FPGA. Related works are presented in Sect. 2. For a better understanding of the proposed method, a mathematical formulation of the problem is given in Sect. 3. Section 4 presents the iterative method for the resolution of linear systems based on GD. The proposed circuit for implementing the algorithm is described in Sect. 5. To present the results, Sect. 6 quantifies the resources used and the operating rate for a case study of the proposed circuit in an Inverse Filtering problem, also known as Channel Equalization. Finally, the conclusions are outlined in the last section.

2 Related Works

The challenge of resolving LES for streaming data with both low latency and high-speed constraints has been pursuit recently, achieving good results for throughput rate around hundred of kHz. In Jiang and Raziei (2017) the authors implement a Gauss-Jordan Elimination (GJE) procedure to solve LES for DCFS in fixed-point arithmetic. This method is non-iterative, reaching the closed solution in few steps, reducing drastically the number of operations. However, closed solution techniques impose some limitations when compared to the GD method proposed in this paper:

- It cannot be used for neither non-square matrix equation nor in application with additive noise. Iterative methods, like the one proposed here, can be interpreted as a Least Square approximation procedure, reaching a wider number of applications.
- That procedure requires the implementation of digital division circuit. This kind of circuitry uses a huge amount of resources and can generate instability in fixed-point circuitry (typical for FPGA) due to the possibility of division by very small numbers.
- The division circuit tends to increase the final costs and reduce the overall throughput rate since it consumes a big amount of hardware resources.

To solve the problem of the instability of using divider circuits in fixed-point arithmetic, the work presented in Xiaofang and Ziavras (2003) implements the LES for DCFS through embedded processor, with floating-point hardware. The usage of embedded processors in FPGA is very attractive for intricate arithmetic operations like LES. In that paper, a LU factorization procedure was implemented, which is a non-iterative method suitable for square matrix only, like the previous mentioned work (Jiang and Raziei 2017). Another obvious constraint of this method is the high latency achieved due to the implementation in software. To increase the speed, the authors propose a parallel processing in a multi-core architecture. As a result, throughput rate of hundreds of kHz may be implemented. However, the usage of several instances of processors with floating-point hardware increases drastically the amount of resources in hardware.

The method proposed in Sun et al. (2008) solves some problems of the works presented above. First, the authors use a general iterative procedure to solve LES, allowing its usage in application with non-square matrix as well. Second, they propose a design of a hybrid arithmetic architecture, where each part of the circuit implements a different and optimized floating-point precision. The applied iterative method is called Iterative Refinement (Golub and Van Loan 1996). This method performs a more accurate update to obtain the direction of the gradient of the next iteration when compared to the GD method proposed here, achieving the solution with less iterations. However, that computation still requires the usage of dividers and more complex circuits. As a consequence, no relevant gain in terms of resources are obtained. Since the focus of that paper is on the LES method itself, details of the hardware design were not presented.

In this work, we propose a LES solver based on GD. This is an iterative procedure being more flexible than closed methods in terms of range of applications. Besides, the GD algorithm is based only in add and multiplication operations, reaching a higher frequency clock and lower latency as compared with previous work. The number of necessary iterations of the GD is higher than the one from closed solution. However, in this work we design a circuitry whose the number of iterations does not affect the latency time, which is fixed in two times the size of the linear system (in term of clock cycles). This is an important characteristic when the main goal is to keep a high throughput rate regardless the achieved precision. By not affecting the latency time, the number of iterations will affect the amount of parallel processing. Therefore, this kind of implementation is very suitable for FPGA.

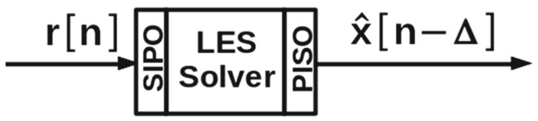

Fig. 1 Free-running circuit representing the proposed LES solution system. The SIPO and PISO blocks adapt the continuous data-flow to the internal iterative circuit

3 Formulation of the Problem

Equation 1 presents a LES in its matrix form, where the matrix $\mathbf{H}$ (of dimension $N \times M$) and the vector $\mathbf{r}$ (of dimension $N \times 1$) are known and one wants to obtain the vector $\mathbf{x}$ (of dimension $M \times 1$).

$$\mathbf{Hx} = \mathbf{r} \tag{1}$$

To implement a circuit capable of solving this problem in DCFS, the observed signal (vector $\mathbf{r}$) must enter sequentially and synchronously with the clock of the system, as the discrete signal $r[n]$ shown in Fig. 1. Here the n variable is an integer comprising the clock index. In addition, these data are uninterrupted, that is, after receiving the last sample of a vector $\mathbf{r}$, the next sample refers to the first element of the next vector $\mathbf{r}$ to be computed without interruption.

The output of the proposed system presents the vector solution $\hat{\mathbf{x}}$, in sequence, also synchronously with the clock of the system. At this point, it is important to note that if the matrix $\mathbf{H}$ is not square, as $M < N$, the system must be able to send the necessary number of null samples between the windows, to allow the correct data-flow. In case $N < M$, it is the task of the previous system to provide the amount of zeros needed to send the vector $\mathbf{r}$.

Another characteristic of the proposed system is that it has a Δ delay between the first sample that enters and the first sample that exits the system. This is necessary because the solver system needs a given time to solve the current LES. The value of this delay will be discussed in Sect. 5. The PISO (Parallel Input to Serial Output) and SIFO (Serial Input to Parallel Output) blocks are responsible for adapting the sequential data-flow to the internal LES solver circuit. In other words, these blocks make iterative processing transparent to continuous data-flow and will be further detailed in Sect. 5.

4 The Gradient Descent Method

Gradient Descent is an iterative numerical method of optimization that uses local gradient information to find the local minimum of a function, always searching in the opposite (negative) direction to the gradient (Luenberger and Ye 2015). For the resolution of LES with this method, it is important to solve Equation 1, where $\mathbf{H}$ and $\mathbf{r}$ are known. The basic idea of this method is based on finding a quadratic function minimizer

$$J(\mathbf{x}) = (\mathbf{r} - \mathbf{Hx})^T(\mathbf{r} - \mathbf{Hx}) \tag{2}$$

Starting from an initial guessing value, $\mathbf{x}_0$, this method moves in the direction of the minimum, that is, in the opposite direction of the gradient $\nabla J(\mathbf{x})$ in small steps, so that at the i-th step, we have

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mu\nabla J(\mathbf{x}_i) \tag{3}$$

The step size is dictated by a constant μ to be determined experimentally. A way to find the best μ value is using Eq. 4 in each iteration of the algorithm.

$$\mu = \frac{\varphi^T\varphi}{\varphi^T(H^TH)\varphi} \tag{4}$$

where φ is the residual vector $\mathbf{r} - \mathbf{Hx}$. The dynamical μ value is found minimizing Eq. 2 with respect to the μ value, after combining Eqs. 2 and 3 (Bertsekas 1999).

After some transformations in Eq. 3, one reaches

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \mu(\mathbf{H}^T\mathbf{r} - \mathbf{Ax}_i) \tag{5}$$

The pseudo-code of the GD algorithm implemented in this work can be seeing in Algorithm 1, where the calculation of $\mathbf{A} = \mathbf{H}^T\mathbf{H}$ is outlined in line 2. The iteration is performed between the lines 6 and 17. The content between parentheses of Eq. 5 is executed at the line 12. Equation 4 is performed at the line 13 (at this algorithm it was considered the matrix $\mathbf{A}$), and the first part of Eq. 5 is found in the line 14.

In the next section, Algorithm 1 will be translated as circuitry modules using Verilog Hardware Description Language (Palnitkar 2003).

5 Implementation

A digital circuit, implementing the proposed Algorithm 1, was designed, using Verilog. The design focused on takes advantage of the parallel processing structure found in FPGA's architecture. Most of the matrix computation are implemented in parallel, using logic elements. Only the matrix computation results are registered. Hence, only one clock cycle is necessary for this computation. This characteristic is crucial for the success of the proposed circuit, as it will be shown below.

In the block diagram of Fig. 2, it is possible to identify a system with 3 pipelines stages (Oda and Oyama 1996), separated by two register banks, where:

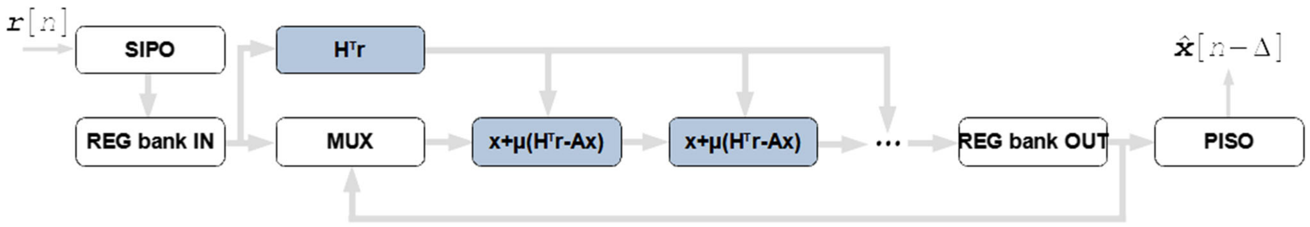

Fig. 2 Block diagram of the proposed circuit. This circuit operates synchronously with the clock of the input signal $r[n]$

Algorithm 1: GRADIENT DESCENDENT

```
Data: Size of the sample, size of the window,
      number of iterations, matrix H and
      signal r
Result: signal $\hat{x}$
1  begin
2      $A \leftarrow H^T H$
3      $W \leftarrow$ initialize
4      $S \leftarrow$ initialize
5      $\hat{x} \leftarrow$ initialize
6      for $it \leftarrow (it + 1)$ to $quantOfSamples$ do
7          $step \leftarrow ((it*window)-(window-1))$
8          $W \leftarrow r(step : step + (window - 1))$
9          $S \leftarrow W^T$
10         $H^T r \leftarrow H^T W^T$
11         for $iti \leftarrow (iti + 1)$ to $numOfIterat$ do
12             $B \leftarrow H^T r - (AS)$
13             $\mu \leftarrow B^T B / B^T AB$
14             $S \leftarrow (S + \mu B)$
15         end
16         $\hat{x}(step : step + (window - 1)) \leftarrow S$
17     end
18 end
19 return $\hat{x}$
```

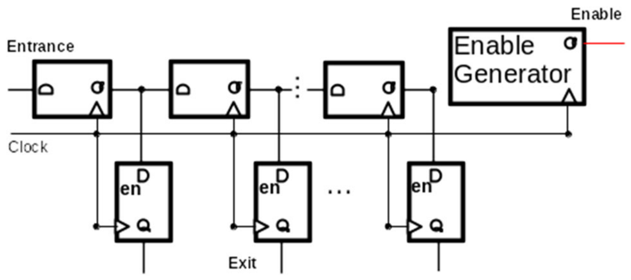

Fig. 3 SIPO circuit and input register bank. An enable signal is generated every N clock cycles to copy the data from the shift register bank into a parallel register bank

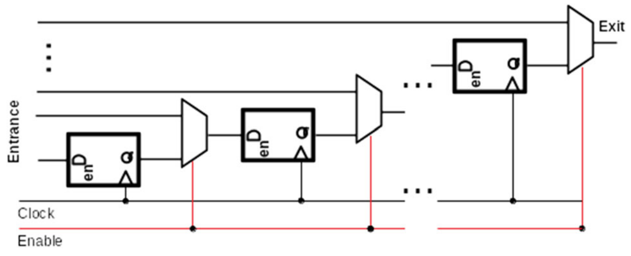

Fig. 4 PISO circuit. Comprised of displacement recorders with load control. To every N clock cycles, the load signal is triggered, loading the data stored in the parallel output register bank (not shown in the figure) into the PISO registers

1. The first pipeline stage is identified as the Serial-Input Parallel-Output (SIPO) circuit, as shown in Fig. 3. This stage assembles a window with N samples (the same size of the vector $\mathbf{r}$). It is identified in the figure where we can see the windowing of the samples that comes from the input. When a window is complete, it is copied to a register bank, to be used by the second stage, while a new window is stored.
2. The second pipeline stage encompasses all the blocks between the incoming and outgoing register banks, shown in the schema of Fig. 2. Among these blocks, the ones responsible for performing the matrix calculations are highlighted. These blocks perform their calculations in a combinational way, in fixed point (Padgett and Anderson 2009) arithmetic and use most of the logical resources of the proposed system. The result is updated in the output register bank at each clock cycle.
3. The third and last stage is performed by the Parallel-Input Serial-Output (PISO) circuit, shown in Fig. 4. This circuit copies the result stored in the output register every N clock cycle (to be synchronized with the size of the vector $\mathbf{r}$ and the size of the window samples). This is also responsible for making the data available in a serial way in the output of the system.

5.1 System Operation

Just after the first N clock cycles, which is when a window in the SIPO register is ready, the multiplexer block is switched to the data coming from the input register bank, initializing the algorithm (initial value of vector **x** as $\mathbf{x}_0$) with the vector **r**. This signal propagates through the internal blocks of the second pipeline stage, which perform the calculations of one iteration of the algorithm. The result is stored in the output register at the rising edge of the next clock. For the very next clock, the multiplexer is switched to the data temporarily stored in the output register database, thus performing new iterations at each clock cycle. This feedback occurs during $N - 1$ consecutive clock cycles.

The second pipeline, therefore, has a total of N clock cycles to execute the iterations of the GD algorithm, before a new window been delivered by the SIPO block. In this way, if a single iteration block is used, a total of N iterations can be performed. However, assuming that k blocks of GD are used in cascade, a total of kN iterations will be performed.

From the above, it is possible to calculate the latency of the system (value of Δ). This is the time needed for the SIPO circuit to store a N samples window, plus the time for the second pipeline stage to execute the k iterations N times, i.e., $\Delta = 2N$. As it can be seen, the circuit delay is fixed and independent of the number of iterations. Besides, the number of iterations is always multiple of N.

To reduce the number of circuits, the matrix calculation $\mathbf{H}^T\mathbf{r}$ is performed only once and the resulting vector is used in the GD interaction blocks, since this vector is constant for each input signal window.

5.2 GD Iteration Block

The combinational circuit that performs an iteration of the GD algorithm is shown in Fig. 5.

The data-flow in this circuit is described below:

1. The signal coming from MUX is multiplied by the matrix $\mathbf{A} = \mathbf{H}^T\mathbf{H}$.
2. To the signal coming from the constant vector $\mathbf{H}^T\mathbf{r}$, it is applied a bit shift to the left. This bit alignment is necessary for the next computation.
3. The difference between the results of the two previous circuits is calculated.
4. This difference is shifted by p bits to the right, the equivalent of multiplying by the value of μ, i.e., $\mu = 2^{-p}$. This strategy avoids the use of an additional multiplier. The value of p is dependent on the application.
5. After the previous computation, the result is added to the signal coming from the MUX in order to complete the iteration calculus.
6. Finally, a final alignment is performed, to ensure the same bit representation of the input. This last step ensures that multiple iteration blocks can be cascaded together, allowing multiple iterations to be performed in a single system clock cycle.

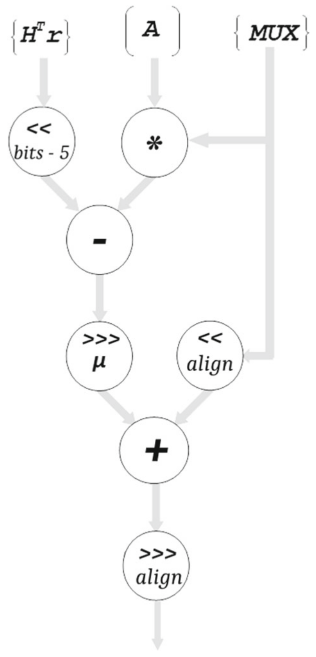

Fig. 5 Block diagram of the circuit that executes a GD iteration. This circuit is fully combinational

6 Case Study—Channel Equalization

To demonstrate the use of the proposed system and quantify the results with a practical case, the LES solution system is applied to a problem of Inverse Filtering (Fritzell 1992) also known as Channel Equalization. However, this method could be used virtually in all kinds of problems that needs inverse-filtering (Leung et al. 2007; Ezri and Tsodik 2011). Figure 7 presents the problem in question. For sake of simplicity, in this example the message to be transmitted is codified in the amplitude of samples of the $x[n]$ sequence, as indicated in Fig. 7. After passing through a physical channel, like the air, with known $h[n]$ impulse response, the receiver acquires the $r[n]$ sequence. In this example, all the process of transforming the message into an analog signal, the transmission and reception by antennas and the signal digitization on the receiver are omitted and incorporated on the $h[n]$ impulse response.

In order to recover the original message, a very common employed procedure is to implement a digital filter on the receiver input (Haykin 1996). The goal is to implement a filter whose impulse response removes the contribution of $h[n]$. Figure 6 shows the overall block diagram for the signal path, since its transmission up to the recovering on the receptor. Generally, it is not possible to implement an exact causal-stable digital inverse filter and Finite Impulse Response (FIR) filters are used to approximate the task (Mitra 2010), using a given optimization criterion. FIR filters are suitable for FPGA

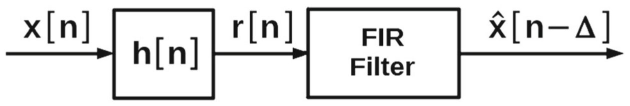

Fig. 6 Scheme of the use of a FIR filter to recover the $x[n]$ sequence

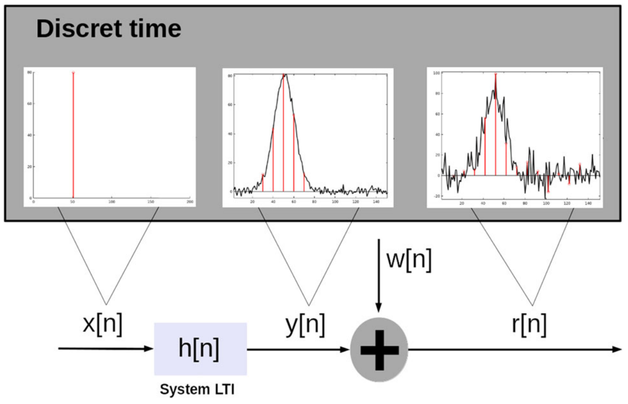

Fig. 7 Example of a simple message $x[n]$ transmitted by a channel with impulse response $h[n]$. $r[n]$ represents the digital sequence acquired by the receiver

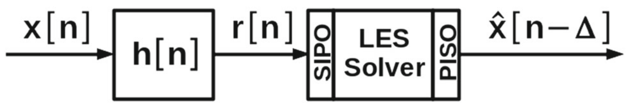

Fig. 8 Example of the use of the proposed circuit in system equalization. The $x[n]$ signal is recovered with the smallest possible mean squared error and a delay of $\Delta = 2N$

implementation and are, so far, the most used implementation procedure.

The proposed free-running LES solver can be used to remove the contribution of $h[n]$ on the receiver input, even for high-speed communication. The authors have not found counterparts of this technique on literature, so far. Besides, for practical cases, the noise information is amplified during the filter inversion process, using FIR filters. On the other hand, iterative processes, like the one proposed here, copes with modern Sparse Representation Theory, where the noise can be shrinkage at each iteration, increasing the final performance (Elad 2010). Therefore, the goal is to implement a LES solver on the receptor input in order to recover a delayed version of the transmitted sequence, as shown on the scheme in Fig. 8, with higher performance than straightforward FIR filters.

Figure 9 shows the $h[n]$ sequence used in this example. A Gaussian-shaped pulse was chosen, where it is possible to identify the 7 used samples. The pulse used was normalized (unit amplitude), so that the amplitude comprises the value of the central sample (without noise).

The relationship between the $x[n]$ input and the $r[n]$ output, in this case (Oppenheim 2010), is given by the Convolution Equation:

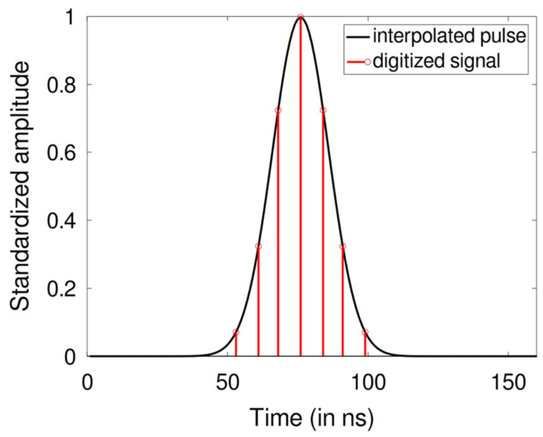

Fig. 9 Channel impulse response $h[n]$ of the transmission medium

$$r[n] = \sum_{l=0}^{6} h[n-l]x[n] \tag{6}$$

We can also express the convolution as a matrix multiplication, using the technique described in Xu et al. (1994). To illustrate it, considers a window formed by a sequence of only five samples of $x[n]$ ($\mathbf{x} = [x_0\ x_2\ x_3\ x_4]$). Since the given channel impulse response $h[n]$ is seven samples wide ($\mathbf{h} = [h_0\ h_1\ h_2\ h_3\ h_4\ h_5\ h_6]$), according to Equation 6, the length of the resulting $r[n]$ sequence is $5 + 7 - 1 = 11$ samples. This can be better depicted using the matrix formulation for finite sequences convolution

$$\begin{bmatrix} h_0 & 0 & 0 & 0 & 0 \\ h_1 & h_0 & 0 & 0 & 0 \\ h_2 & h_1 & h_0 & 0 & 0 \\ h_3 & h_2 & h_1 & h_0 & 0 \\ h_4 & h_3 & h_2 & h_1 & h_0 \\ h_5 & h_4 & h_3 & h_2 & h_1 \\ h_6 & h_5 & h_4 & h_3 & h_2 \\ 0 & h_6 & h_5 & h_4 & h_3 \\ 0 & 0 & h_6 & h_5 & h_4 \\ 0 & 0 & 0 & h_6 & h_5 \\ 0 & 0 & 0 & 0 & h_6 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \end{bmatrix} \tag{7}$$

Equations 7 and 1 are the same. In other words, it is possible to implement a LES solver to perform a Channel Equalization procedure when the message is transmitted in fixed window length. Thus, the goal is to design an inverse linear system in order to recover the $x[n]$ signal with the least possible error. In this example, the recovered signal is named $\hat{x}[n]$.

Assuming that the information in $x[n]$ is encoded in fixed size windows $M = 48$ with 7 guard samples between windows, to avoid inter-symbolic interference (Lee and

Messerschmitt 1988) (due to the spread of information after the convolution with $h[n]$), we have that $N = M+7-1 = 54$. In this case, the matrix model given by Equation 1 can be used to represent the convolution between the input and output sequence of a given window. To do this, the matrix $\mathbf{H}$ contains, in its columns, shifted versions of the $h[n]$ with zero padding. At this point, it is important to point out that because the matrix $\mathbf{H}$ has fewer rows than columns ($M < N$), the LES in question does not have a single trivial solution and the GD method will result in the solution with the lowest Root Mean Square error (RMS) between $x[n]$ and $\hat{x}[n]$ (Kelley 1995), as determined by the cost function given by Equation 2.

7 Experimental Results

This section discusses the experimental results of the LES resolution circuitry. A Python tool was developed (high-level programming language with a wide range of resources and libraries (McKinney 2017)) that generates all method implementation codes, based on user-defined parameters, such as quantization, window size and number of GD iteration blocks in cascade. The tool generates code in Verilog (Palnitkar 2003) (hardware description language (Botros 2015)) for circuit implementation in programmable logic devices. The circuits can be synthesized in devices through the development tools of each manufacturer, such as Altera/Intel's Quartus II (Parab et al. 2018) and Xilinx's Vivado (Churiwala 2016). The high-level codes simulate the fixed point construction of the Verilog equivalent and also implement floating point algorithms for performance comparisons. This Python tool developed here is a high-level synthesis (HLS) (Choi et al. 2016).

7.1 Database

To test the proposed circuit, a signal was generated consisting of 1,820 sample windows, where each window is composed of a sequence of 48 samples containing random values. This sequence corresponds to a single digital symbol to be transmitted. To form the final $x[n]$ input sequence, the 1,820 windows were concatenated, adding another 7 samples with zero value between them. Finally, to this signal is added a white Gaussian noise (Trees 2013) with 1 ADC count standard deviation, representing the electronic noise of the system.

The $x[n]$ signal generated is then convoluted with the $h[n]$ signal shown in Fig. 9, generating the $r[n]$ signal. The latest should be interpreted as output samples from a free-running ADC in the receiver. In this way, the $r[n]$ sequence is then modified to a representation of integers (Barzee 2014) of 10 bits. With the $r[n]$ quantized signal, it was submitted as input to the proposed circuit. Figure 10 shows a piece

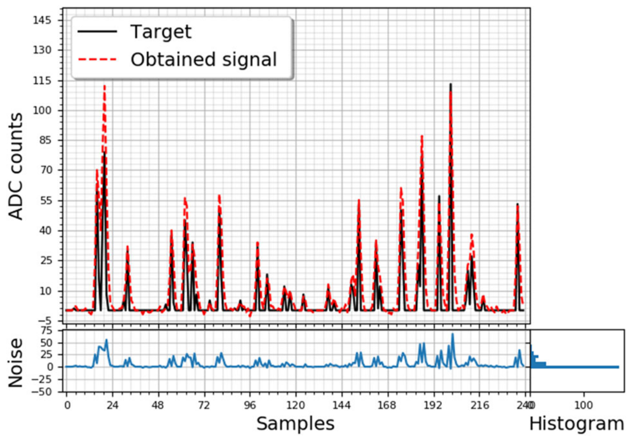

Fig. 10 Samples of the signals before (Target) and after (Obtained Signal) passing through the $h[n]$ channel (sequence $x[n]$ and $r[n]$, respectively). The bottom graph shows the error between the output and input signals, point to point

of the simulated $x[n]$ sequence (Target) and its $r[n]$ output (Obtained Signal) after passing the $h[n]$ linear system. The two sequences were aligned in order to quantify the difference between them, given in the graph at the bottom of this figure, indicating the noise introduced by the transmission channel.

7.2 Quantization of the GD Iteration Circuit

Aiming at an implementation in FPGA, the matrix calculations of the main circuit, shown in Fig. 5, are all executed in fixed-point. The quantization process depends on the conditioning of the input data and the accuracy required in the output data, being a specific characteristic of the problem in which the LES resolution circuit will be applied. In the Inverse Filtering problem, described in this case study, it is suggested a sufficient quantization to implement the algorithm capable of giving an error of less than 1% in relation to the same algorithm implemented in floating point.

In the study of the quantization to be performed, the fixed point version of the method was exposed to a total of 216 variations in 3 evaluated metrics, summarized in Table 1.

After testing all variations, it was concluded that 7 bits for the integer part and 10 bits for the fractional part, for the example in question, were sufficient to maintain an error within 1% in relation to the same method implemented in floating point, regardless of the number of blocks of iterations GD cascading (1, 2 or 3 blocks).

7.3 The μ Constant Determination

Once the quantization to be used has been determined, it is necessary to determine the step size of the GD algorithm, i.e., the constant μ in Equation 5. As described in Sect. 4, initially

Table 1 Set of the tested possibilities for circuit construction configurations

Metric	Start	End	Step	Variations
Number of Blocks of GD iterations	1	3	1	3
Integer part of binary representation (*bits*)	5	17	1	12
Fractional part of binary representation (*bits*)	5	11	1	6
Total				**216**

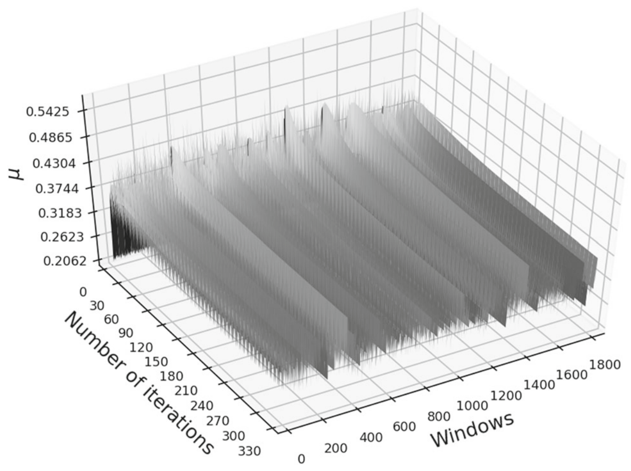

Fig. 11 All values of μ generated dynamically

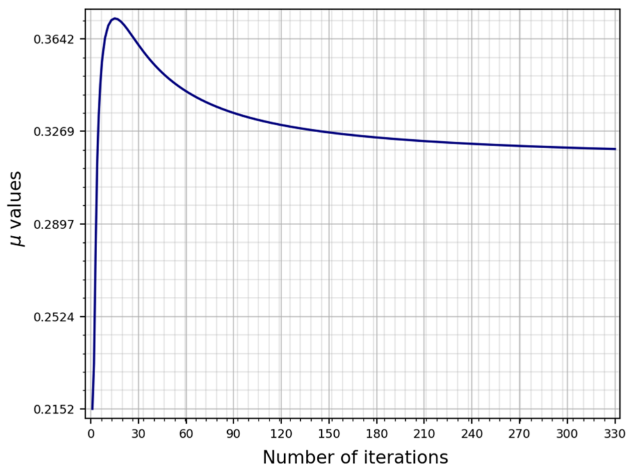

Fig. 12 Mean values of the dynamical μ as a function of the iteration index

Equation 4 can be used to determine an optimal value dynamically. However, in order to simplify the implementation in hardware, it should be a constant according to $\mu = 2^{-p}$. Figure 11 shows the optimal values for μ, where it was plotted each value generated in each one of 1,820 sample windows. A total of 330 iterations was used in the calculation to each window.

The mean value of μ as a function of the iteration index is calculated and plotted in Fig. 12, where one can identify a tendency of a value of μ between 0.21 and 0.37, with a mean near to 0.33. Hence, it is necessary to evaluate the performance of the signal reconstruction for p varying between 1 and 3.

During the developing of this work, we considered the Root Mean Square (RMS) error between the target signal and the reconstructed signal ($x[n]$ and $\hat{x}[n]$ respectively), as a function of the iteration index, for the three different values of μ (Figure 13). We noted that for $\mu = 0.5$ ($p = 1$, or $\mu = 2^{-1}$) the algorithm diverges, showing that a smaller value should be used. For $\mu = 0.25$ ($p = 2$) and $\mu = 0.125$ ($p = 3$), the algorithm converges to the lowest RMS, whereas with $\mu = 0.25$, it converges faster and get values very near the values obtained with the μ calculated using the Equation 4. Therefore, the $\mu = 0.25$ was chosen as parameter in the circuit.

Figure 14 shows the sequence reconstruction presented in Fig. 10, for 330 iterations, with $\mu = 0.25$. The signals were

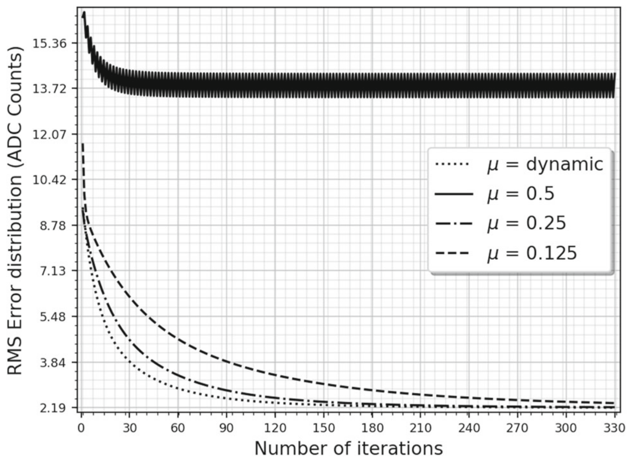

Fig. 13 RMS error as a function of the iteration index for dynamical and three different fixed values of μ

realigned to promote the calculation of the point-to-point error, shown in the graph below this figure. It can be seen, visually, a good recovery of the input signal. A quantitative analysis is given below.

7.4 Implementation Performance

The proposal of this section is to describe the results, making possible to the reader compare them with the results obtained in previous works about LES. However, each work presents

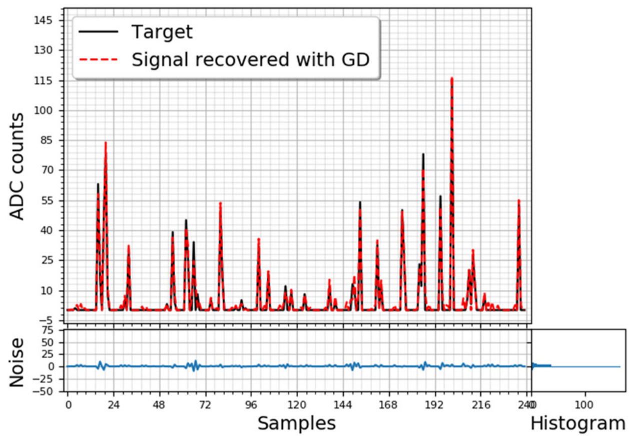

Fig. 14 Example of a generate ($x[n]$) and recovered ($\hat{x}[n]$) sequences, using the proposed technique

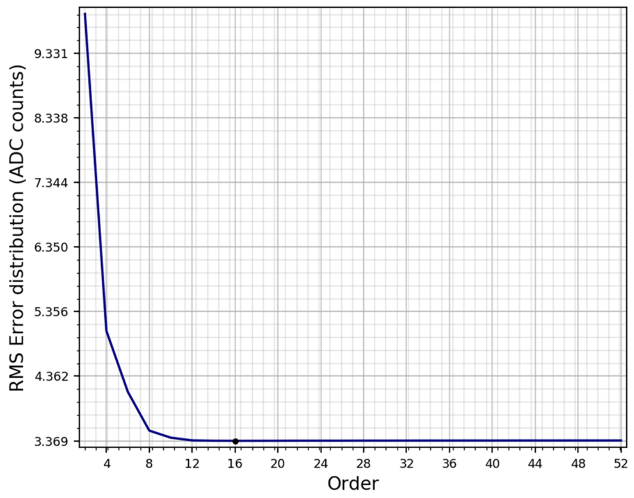

Fig. 15 Relation between the FIR filter order and the signal reconstruction performance

their results using a different point of view, as could be seen in Yi and Li (2015), where it was necessary only 1,324 clock cycles to perform a Gaussian elimination, when solving LES with matrix size 50 × 50. In Salah and AbdelSalam (2017), it was necessary the use of 27,000 LUTs and 64 DSP blocks to implement their encryption method, running at $300MHz$. As it can be found in Sudarsanam and Aravind (2005), in a seven pipeline stages, they could solve a LES, with a throughput where every clock 16 values was set, in a clock width of $32ns$ or $31.25MHz$.

Because each work uses a different FPGA board, it makes more difficult to compare this work with previous ones, that also implemented LES solving in FPGA. However, it was not identified a work for streaming processing solving LES with throughput higher than dozens of kHz. The proposed circuit was synthesized in Vivado (Churiwala 2016), using a Xilinx FPGA from the Kintex UltraScale+ KCU116, the xcku5p-ffvb676-2-e (UltraScale 2019), with 216,960 LUTs and 433,920 FFs.

Table 2 presents a summary of the performance results. The results are presented for k ranging from 1 to 6. Thus, it was possible to perform a number of 55 to 330 iterations, in multiples of 55. However, it is possible to observe that, with about 165 iterations, the algorithm is already very close to its maximum performance, in terms of accuracy of the result. This can also be measured by the last line of this table.

As for the amount of logical resources (LUT and FF), it may use about 89.72% of the resources of the FPGA for $k = 6$ (330 iterations), which is an extreme case. On the other hand, for $k = 2$ (110 iterations) it consumes just 29.71% of the resources still reaching a satisfactory result. However, this is an example with a large LES, with 48 independent variables, exactly to show that the circuit can be implemented, even for large matrices.

Even being a complex circuit, which demands a high cost of logical resources, it was able to operate at tens of MHz and can be operated with more than $30MHz$ of real throughput data (clock of $r[n]$), even for 275 iterations.

In order to allow a comparison with a method commonly used in channel equalization, a linear equalizer was designed, whose inverse filter is approximated by a FIR filter type structure. It is the most widely used method for this purpose (Diniz 2006), given its simple implementation in FPGA. Supposing a target operation at 40MHz of signal flow, this time constraint was applied to the Vivado software. Table 2 indicates the circumstances where this requirement has been met.

To proceed with the development of a FIR filter that could be used as a parameter of comparison with the GD algorithm here implemented, first it was necessary to find the filter order. The relationship between the RMS error and the order of the FIR filter is shown in Fig. 15. On this figure, it is possible to identify that the filter reaches its smaller RMS error with an order of 16. Therefore, this will be the filter order used on the implementation.

Table 2 also presents resources used and the efficiency of this method. It can be seen that, although the FIR filter structure uses fewer resources, the deconvolution procedure by direct resolution of LES presents a more reliable reconstruction of the input signal. The proposed circuit, therefore, opens a new perspective of equalizer design based on direct resolution of LES that can be implemented for high acquisition rate in free-running environments, using modern FPGAs.

Figure 16 presents a complementary information related to Table 2, concerning performance comparison between FIR filter and direct LES solver implementation. The GD algorithm starts to outperform the FIR filter method for a number of iterations above 110.

Concerning comparison with closed related works (Jiang and Raziei 2017; Xiaofang and Ziavras 2003; Sun et al. 2008), the latency reduction on those papers is not so

Table 2 Table with the results, obtained from the implementation of the proposed method

Fixed-point method, with $\mu = 0.25$							
Logical Elements	FIR	GD Iterations					
		55	110	165	220	275	330
LUT	1,447	33,775	65,470	97,293	140,557	160,964	194,654
FF	731	2721	2721	2721	2799	2790	2793
DSP	125	308	376	444	512	580	648
Fmax (MHz)	56.76	77.32	59.58	47.31	40.00	31.35	24.85
Power (W)	0.495	0.867	1.242	1.612	2.121	2.458	2.839
RMS (ADC counts)	3.3705	3.4938	2.6121	2.3517	2.2589	2.2207	2.2032

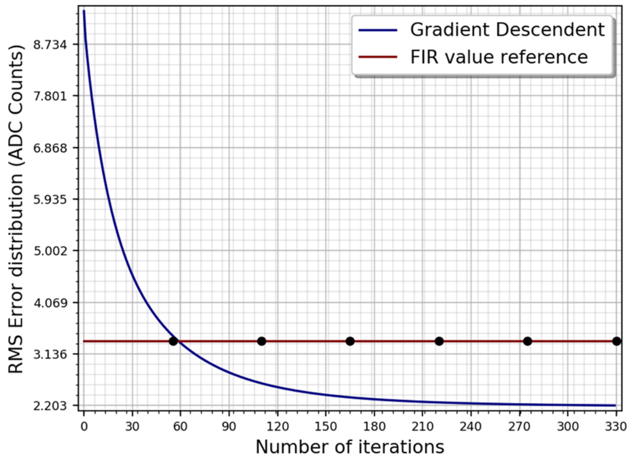

Fig. 16 Comparison between GD and FIR filter as a function of iteration index. For the FIR case, this graphic is constant since it is not a iterative method

expressive when compared to the methods based on batch algorithms (Binotto et al. 2010; Khun et al. 2015; Salah and AbdelSalam 2017). In Jiang and Raziei (2017) for instance, the latency is of 5 microseconds in a Virtex FPGA at 200 MHz of primary clock (throughput of kHz). In the method proposed here, the latency, in terms of number of clock cycles, is fixed as twice the size of the system, regardless the number of iterations used.

In Sun et al. (2008), the authors don't inform the hardware resources, since it is not the focus of that paper. Concerning the latency time, it varies according to the order of the linear system, as expected. A 24x24 matrix, for instance, is processed in 500 microseconds (2kHz), while a 100x100 matrix is executed in 20 milliseconds.

8 Conclusion

This paper presented the development of a circuit for LES solution, based on the GD method, to be used in DCFS. The system works synchronously and only with the clock of the input data. The system latency is fixed at twice the size of the input signal window, regardless of the number of iterations desired. Such circuit is of generic use and no work with the same characteristics was identified by the authors. Therefore, it can be of great importance for systems that operate in continuous high speed data-flow, such as telecommunications, nuclear instrumentation and radars, for example.

Since it is an iterative algorithm with matrix calculations, the greatest challenge was its adaptation to DCFS operation. The use of a large block of combined circuits to perform all the calculations of a GD iteration in parallel, using the great granularity present in modern FPGAs, made this implementation possible. To allow the correct data-flow, without pipeline breakage, SIPO and PISO circuits were used at the ends.

An important feature of the developed circuit was the ability to concatenate resolution blocks of a GD iteration. For this, the same bit quantization was used at all points of the circuit, allowing this concatenation to be done in a simplified way. This feature makes the latency of the system constant and independent of the number of iterations chosen.

A case study to solve a LES with 48 variables and 55 equations demonstrated the good performance of the system. Even for a matrix of this dimension (55×48) the system is able to perform 330 iterations, operating free-running with a clock above $20MHz$ in a medium sized commercial FPGA.

Acknowledgements The authors would like to thank CNPq, FAPEMIG (Brazil) and CERN (Switzerland) for their support to this work.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Akidau, T., Chernyak, S., & Lax, R. (2018). *Streaming systems: The what where when and how of large-scale data processing*. USA: OReilly Media Incorporated.

Barzee, R. A. (2014). *Really understand binary*. USA: Maia LLC.

Bertsekas, D. P. (1999). *Nonlinear programming*. USA: Athena Scientific.

Binotto, A. P. D., Daniel, C., Weber, D., Kuijper, A., Stork, A., Pereira, C., & Fellner, D. (2010). Iterative sle solvers over a cpu-gpu platform. In 2010 IEEE 12th International Conference on High Performance Computing and Communications (HPCC), pp 305–313.

Botros, N. (2015). Hdl with Digital Design. Mercury Learning & Information.

Choi, J., Lian, R. L., Brown, S., & Anderson, J. (2016). A unified software approach to specify pipeline and spatial parallelism in fpga hardware. In: *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. Los Alamitos, CA, USA: IEEE Computer Society.

Churiwala, S. (2016). *Designing with Xilinx FPGAs: Using Vivado* (1st ed.). Berlin: Springer Publishing Company Incorporated.

Diniz, P. (2006). *Adaptive filtering?: algorithms and practical implementation*. New York: Springer.

Diniz, P. (2010). *Digital signal processing?: system analysis and design*. New York: Cambridge University Press.

Elad, M. (2010). *Sparse and redundant representations*. New York: Springer New York.

Ezri, D., & Tsodik, G. (2011). New time domain equalizer for multitone vehicle to vehicle communications. In 2011 IEEE International Conference on Microwaves, Communications, Antennas and Electronic Systems.

Fedasyuk, D., Serdyuk, P., & Semchyshyn, Y. (2009). Hierarchical distribution of high dimensional block-banded sle solving. In 2009 10th International Conference - The Experience of Designing and Application of CAD Systems in Microelectronics, pp 292–295.

Fritzell, B. (1992). Inverse filtering. *Journal of Voice*, 6(2), 111–114.

Gerstacker, W. H., Obernosterer, F., Schober, R., Lehmann, A. T., Lampe, A., & Gunreben, P. (2004). Equalization concepts for alamoutis space-time block code. *IEEE Transactions on Communications, 52*(7), 1178–1190.

Golub, G. H., & Van Loan, C. F. (1996). *Matrix computations*. USA: The Johns Hopkins University Press.

Haykin, S. (1996). *Adaptative filter theory*. NJ, USA: Prentice Hall.

Jiang, Z., & Raziei, S. A. (2017). An efficient fpga-based direct linear solver. In 2017 IEEE National Aerospace and Electronics Conference (NAECON), pp 159–166.

Jolliffe, I. (2002). *Principal component analysis*. New York: Springer Verlag.

Kelley, C. T. (1995). Iterative Methods for Linear and Nonlinear Equations. No. 16 in Frontiers in Applied Mathematics, SIAM.

Khun, J., Šimeček, I., & Lórencz, R. (2015). Gpu solver for systems of linear equations with infinite precision. In 2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing.

Kuon, I., Tessier, R., & Rose, J. (2008). Fpga architecture: Survey and challenges. *Found Trends Electron Des Autom, 2*(2), 135–253.

Lee, E. A., & Messerschmitt, D. G. (1988). *Intersymbol Interference* (pp. 289–370). Dordrecht: Springer Netherlands.

Leung, M., Kot, J. S., & Jones, B. B. (2007). Wideband dual polarized line feed development for a cylindrical reflector radio telescope. In 2007 IEEE Antennas and Propagation Society International Symposium, pp 1929–1932.

Luenberger, D. G., & Ye, Y. (2008). *Linear and Nonlinear Programming*. New York: Springer US.

Luenberger, D. G., & Ye, Y. (2015). *Linear and nonlinear programming*. Berlin: Springer Publishing Company Incorporated.

Martinez-Alonso, R., Mino, K., & Torres-Lucio, D. (2010). Communication scheme design for parallel architecture implemented in a fpga for solution of linear equation systems. In 2010 IEEE Electronics, Robotics and Automotive Mechanics Conference, pp 726–730.

McKinney, W. (2017). *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython, NumPy, and IPython* (2nd ed.). USA: OReilly Media.

Mitra, S. K. (2010). *Digital signal processing: A computer-based approach*. New York: McGraw-Hill Higher Education.

Oda, Y., & Oyama, T. (1996). Fast calculation using parallel processing and pipeline processing in power system analysis. *Electr Eng Japan - ELEC ENG JPN, 116*, 85–96.

Oppenheim, A. (2010). *Discrete-time signal processing*. NJ: Pearson.

Padgett, W., & Anderson, D. (2009). *Fixed-Point signal processing*. USA: Morgan & Claypool Publishers LLC.

Palnitkar, S. (2003). *Verilog HDL* (2nd ed.). USA: Prentice Hall.

Parab, J. S., Gad, R. S., & Naik, G. M. (2018). *Getting hands on altera quartus II software* (pp. 19–37). New Delhi: Springer.

Proakis, J. G., & Saleh, M. (2007). *Digital Communications*. Boston: McGraw-Hill.

Saad, Y. (2003). Iterative Methods for Sparse Linear Systems, (2nd edn.). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Salah, K., & AbdelSalam, M. (2017). A comparative analysis between fpga and gpu for solving large numbers of linear equations. In 2017 29th International Conference on Microelectronics (ICM), pp 1–4.

Sudarsanam, A., & Aravind, D. (2005). A fast and efficient fpga-based implementation for solving a system of linear interval equations. In Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005., pp 291–292.

Sun, J., Peterson, G. D., & Storaasli, O. O. (2008). High-performance mixed-precision linear solver for fpgas. *IEEE Transactions on Computers, 57*(12), 1614–1623.

Trees, H. (2013). *Detection, estimation, and modulation theory*. Hoboken: Wiley.

UltraScale. (2019). Ultrascale and ultrascale+ fpgas packaging and pinouts.

Xiaofang, Wang, & Ziavras, S. G. (2003). Parallel direct solution of linear equations on fpga-based machines. In Proceedings International Parallel and Distributed Processing Symposium, pp 8.

Xu, C., Aissaoui, I., & Jacquey, S. (1994). Algebraic analysis of the van cittert iterative method of deconvolution with a general relaxation factor. *Journal of the Optical Society of America A, 11*(11), 2804.

Yang, D., Peterson, G. D., Li, H., & Sun, J. (2009). An fpga implementation for solving least square problem. In 2009 17th IEEE Symposium on Field Programmable Custom Computing Machines, pp 303–306.

Yi, H., & Li, W. (2015). Small fpga implementations for solving systems of linear equations in finite fields. In 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS), pp 561–564.

Yu, Y., Luo, Y., Wang, D., Fu, S., & Xu, M. (2016). Efficient, secure and non-iterative outsourcing of large-scale systems of linear equations. In 2016 IEEE International Conference on Communications (ICC), pp 1–6.