Advances in Radio Science, 3, 325–329, 2005
SRef-ID: 1684-9973/ars/2005-3-325
© Copernicus GmbH 2005

Advances in
Radio Science

An Adiabatic Architecture for Linear Signal Processing

M. Vollmer and J. Götze

University of Dortmund, Information Processing Lab., Germany

Abstract. Using adiabatic CMOS logic instead of the more traditional static CMOS logic can lower the power consumption of a hardware design. However, the characteristic differences between adiabatic and static logic, such as a four-phase clock, have a far reaching influence on the design itself. These influences are investigated in this paper by adapting a systolic array of CORDIC devices to be implemented adiabatically.

We present a means to describe adiabatic logic in VHDL and use it to define the systolic array with precise timing and bit-true calculations. The large pipeline bubbles that occur in a naive version of this array are identified and removed to a large degree. As an example, we demonstrate a parameterization of the CORDIC array that carries out adaptive RLS filtering.

1 Functional Simulation of Adiabatic Logic

Adiabatic logic families such as Positive Adiabatic Logic (PFAL, Vetuli et al., 1996) use voltage ramps in order to charge/discharge the capacitances in an energy efficent way. In contrast, traditional static CMOS loads/unloads the capacitances with steep voltage slopes. In addition to two phases where the clock is *high* or *low*, respectively, adiabatic logic uses two more phases of the same duration where the clock transitions in a ramp (Fig. 1, signals ϕ_1 and ϕ_2). See Fischer et al. (2003) for a more detailed description.

The optimum operating frequency of adiabatic logic tends to be lower than that of static CMOS, leading to the desire for highly parallel designs. We will present such a design for linear signal processing in the sequel.

Due to the way basic functional blocks such as *not*, *and* or larger entities like a half-adder are implemented, they are inherently synchronized with the power clock: they sample their inputs in the rising phase of their clock and their outputs are valid during the immediately following high phase.

When basic blocks are connected, they automatically form a pipeline. For example, when two blocks are connected serially, they form a pipeline that can consume one input every clock-cycle and produces one output every clock-cycle with a delay of two phases. The second block needs to sample its inputs while the outputs from the first block are valid. Thus, its clock needs to be in the rising phase while the clock of the first block is in its high phase. In general, an adiabatic circuit therefore needs to provide four synchronized global power clocks such that in every phase all of the four phase kinds (low, rising, high, falling) are available.

Figure 1 shows this situation for two inverters. It also shows the dual rail encoding that adiabatic logic families use: for every input signal, one also needs to provide the logically inverted signal. For a logic *one*, a signal follows its associated power clock, for a logic *zero*, it stays at ground.

To verify the lower energy dissipation of adiabatic logic as compared to static logic, one needs to perform transient simulations on the transistor and wire level with, e.g., SPICE. When designing larger circuits, like the array of CORDICs in the sequel, it is advantageous to first concentrate only on the functional aspects. This allows a simulation to complete much faster and thus mistakes can be made and corrected more quickly.

We have therefore created a simple set of conventions for VHDL that allow the description of logic blocks that are implicitly synchronized with a four phase clock. Figure 2 shows the simplifications relative to Fig. 1 and Fig. 3 shows simulated wave forms. The dual rail encoding is not modeled, and there is only one global clock net. Signals are valid during two phases since VHDL events happen at phase transitions and the sampled signal must be stable at this point. Nevertheless, the phase relations of two blocks can be observed in simulated waveforms and misalignments can be detected.

Figure 4 shows the VHDL code for a adiabatic inverter and Fig. 5 shows how to instantiate the two inverters of Fig. 2. The `phase` generic of a component aligns it with one of the four phases of the global clock.

Building on this conventions, a parameterizable array of CORDIC cells has been developed that can be programmed to carry out a number of signal processing tasks.

Correspondence to: M. Vollmer
(marius.vollmer@udo.edu)

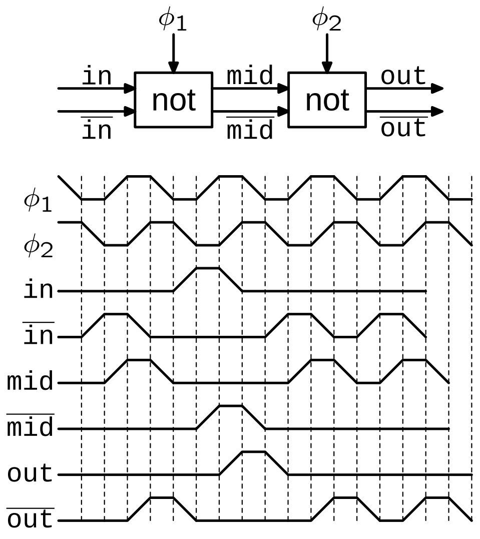


Fig. 1. Two adiabatic inverters.

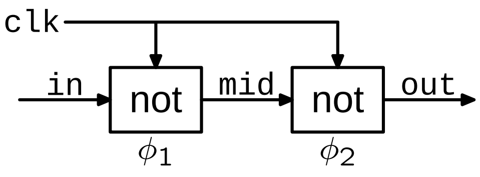


Fig. 2. Two adiabatic inverters in VHDL.

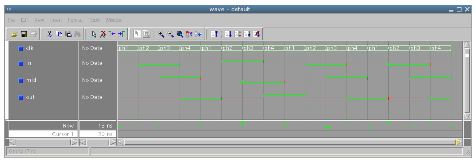


Fig. 3. Simulated wave forms for Fig. 2.

2 Systolic Architecture

The presented architecture is an array of locally connected CORDIC devices that resembles the familiar triangular array for computing a QR decomposition (Haykin, 1996). It is depicted in Fig. 6 together with its inputs and outputs.

Such a device is able to find a transformation $\mathbf{\Theta}$ and apply it to matrices $\mathbf{V}_1$, $\mathbf{V}_2$, $\mathbf{W}_1$, $\mathbf{W}_2$ of real numbers such that

$$\mathbf{\Theta}\begin{bmatrix}\mathbf{V}_1 & \mathbf{V}_2\\ \mathbf{W}_1 & \mathbf{W}_2\end{bmatrix} = \begin{bmatrix}\mathbf{V}'_1 & \mathbf{V}'_2\\ \mathbf{0} & \mathbf{W}'_2\end{bmatrix}. \qquad (1)$$

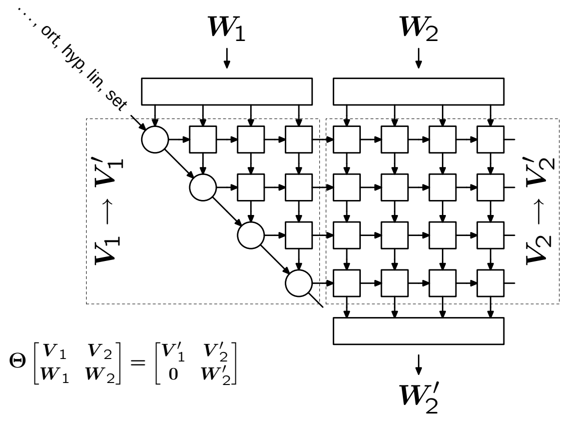
In this equation, $\mathbf{V}_1$, which must be upper triangular, and $\mathbf{V}_2$ represent the values in the internal registers of the architecture. The matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ represent the input values. The transformation $\mathbf{\Theta}$ is chosen such that the $\mathbf{W}_2$ matrix is annihilated. After applying $\mathbf{\Theta}$, the matrices $\mathbf{V}'_1$ (again upper triangular) and $\mathbf{V}'_2$ represent the new values of the internal registers, and the matrix $\mathbf{W}'_2$ represents the outputs.

```
architecture default of adi_inv is
begin

  process
  begin
    o <= 'X';
    loop
      wait on clk;
      if clk = eval_phase (phase) then
        o <= not i;
      elsif clk = reco_phase (phase) then
        o <= 'X';
      end if;
    end loop;
  end process;

end default;
```

Fig. 4. VHDL code for an adiabatic inverter.

```
inv1: entity work.adi_inv
  generic map (
    phase => ph2)
  port map (
    clk => clk,
    i => s_in,
    o => s_mid);

inv2: entity work.adi_inv
  generic map (
    phase => ph2+1)
  port map (
    clk => clk,
    i => s_mid,
    o => s_out);
```

Fig. 5. VHDL code for instantiating two inverters.



W_1
W_2
..., ort, hyp, lin, set
$V_1 \rightarrow V'_1$
$V_2 \rightarrow V'_2$
$\Theta\begin{bmatrix}V_1 & V_2\\ W_1 & W_2\end{bmatrix} = \begin{bmatrix}V'_1 & V'_2\\ 0 & W'_2\end{bmatrix}$
W'_2

Fig. 6. The adiabatic architecture.

The CORDICs can be programmed at run-time to construct transformations $\mathbf{\Theta}$ with different additional properties, see the next section. A circular cell in Fig. 6 represents a *vector* CORDIC: it finds a elementary 2×2 transformation that annihilates a single element of $\mathbf{W}_1$. A square cell represents a *rotation* CORDIC that applies the elementary transformation found by the circular cells in its row.

The CORDIC cells have an internal feedback loop as depicted in Fig. 7. In a way, these loops create the internal

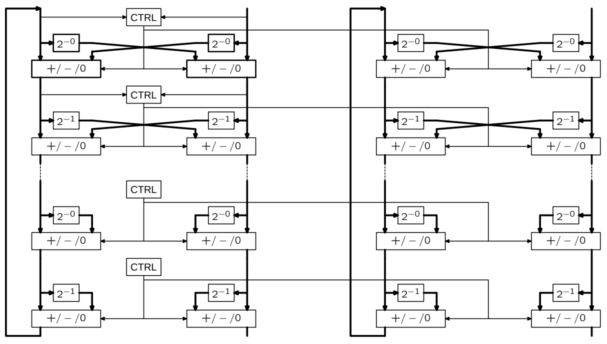


Fig. 7. The CORDIC cells. Left: vector, right: rotation.

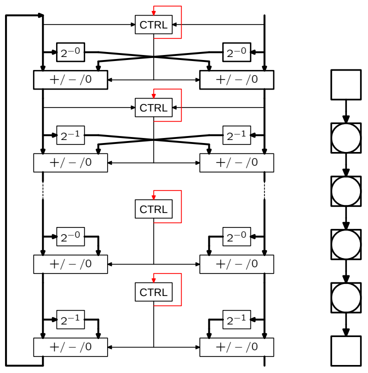


Fig. 8. Area-optimized device.

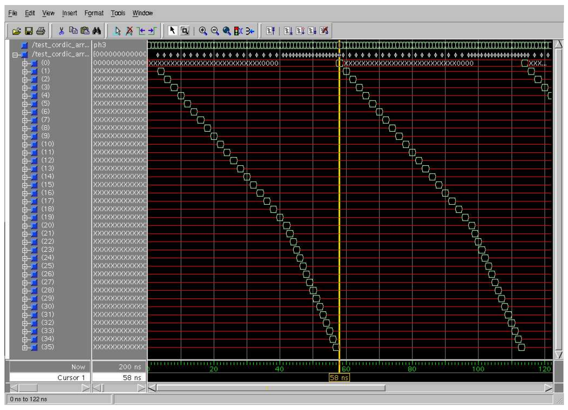


Fig. 9. Pipeline bubbles.

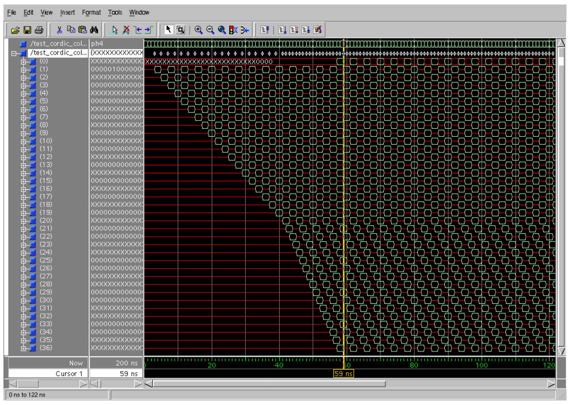


Fig. 10. No pipeline bubbles.

registers that store $\mathbf{V}_1$ and $\mathbf{V}_2$. In a static design, there will be a real register to form the loop, but in an adiabatic design, the adders themselves are pipelined and behave inherently as registers.

The more micro-rotation stages there are in the CORDICs, the longer the pipeline inside each cell becomes. Waiting for the result to be fed back to the input creates large pipeline bubbles. Figure 9 depicts this effect. It shows simulated waveforms of the signals between the stages of one CORDIC. It can be seen that one must wait for a value to slowly ripple through all stages until the next meaningful computation can be started.

To get around this unfortunate situation, one can observe that a given stage in a given CORDIC cell can do the work of its right neighbor during the time it would otherwise sit idle. So instead of distributing the control information to the right, a circular cell can feed this information back to itself and assume the role of its neighbors in subsequent cycles. Those neighbors can be removed from the design. Figure 8 shows the result: a single column of CORDIC devices that can be in both the *vector* and *rotation* modes. Figure 10 confirms that the bubbles have disappeared. The length of the original bubble determines how many columns can be collapsed into one.

3 Linear Transformations for Signal Processing

As mentioned in the previous section, properties of the transformation matrix $\mathbf{\Theta}$ in Eq. (1) can be controlled at run-time such that a number of *modes* are created that the array can be in. The modes and possible applications of them are listed below.

– In the *orthogonal* mode, $\mathbf{\Theta}$ is chosen to be orthogonal, such that the following relationships can be established:

$$\mathbf{V}_1'^H\mathbf{V}_1' = \mathbf{V}_1^H\mathbf{V}_1 + \mathbf{W}_1^H\mathbf{W}_1$$

and

$$\mathbf{V}_1'^H\mathbf{V}_2' = \mathbf{V}_1^H\mathbf{V}_2 + \mathbf{W}_1^H\mathbf{W}_2. \qquad (2)$$

Thus, this mode can be used to carry out a QR decomposition $\mathbf{X}{=}\mathbf{QR}$ of an arbitrary matrix by letting $\mathbf{V}_1{=}\mathbf{0}$ and $\mathbf{W}_1{=}\mathbf{X}$. Then we will find $\mathbf{V}_1'{=}\mathbf{R}$. Additionally, we will find $\mathbf{V}_2'{=}\mathbf{Q}^H\mathbf{W}_2$ which allows us to solve the least squares problem $\min_{\boldsymbol{w}} \|\mathbf{X}\boldsymbol{w}-\boldsymbol{y}\|$ by setting $\mathbf{W}_2{=}\boldsymbol{y}$ and using a subsequent transformation in the *linear* mode (see below).

This mode can also be used to perform a QR updating step, which is most concisely expressed as starting from an upper triangular $\mathbf{R}_1$ such that $\mathbf{R}_1^H\mathbf{R}_1{=}\mathbf{X}_1^H\mathbf{X}_1$ and efficiently finding an upper triangular $\mathbf{R}_2$ such that $\mathbf{R}_2^H\mathbf{R}_2{=}\mathbf{X}_1^H\mathbf{X}_1{+}\mathbf{X}_2^H\mathbf{X}_2$. This can be achieved by letting $\mathbf{V}_1{=}\mathbf{R}_1$ and $\mathbf{W}_1{=}\mathbf{X}_2$, leading to $\mathbf{V}_1'{=}\mathbf{R}_2$. $\mathbf{V}_2$ and $\mathbf{W}_2$ can be used in the same manner as above to update the right hand side such that the solution to a least-squares problem can be updated.

Note that the QR updating step can be used repeatedly to compute an upper triangular $\mathbf{R}$ such that $\mathbf{R}^H\mathbf{R}{=}\sum_i \mathbf{X}_i^H\mathbf{X}_i$ without having to access the internal registers, apart from the initialization $\mathbf{V}_1{=}\mathbf{0}$. In fact, the device effectively computes the QR decomposition of $\mathbf{X}$ by successively updating the solution one row at a time until all rows of $\mathbf{X}$ have been accounted for.

Channel estimation, channel equalization, data detectors and adaptive filters can use the methods mentioned above, for example.

– In the *linear* mode, $\mathbf{\Theta}$ has the form

$$\mathbf{\Theta} = \begin{bmatrix} \mathbf{I} & \\ \mathbf{\Delta} & \mathbf{I} \end{bmatrix} \tag{3}$$

leading to the computation of the *Schur Complement*

$$\mathbf{W}_2' = \mathbf{W}_2 - \mathbf{W}_1\mathbf{V}_1^{-1}\mathbf{V}_2.$$

This can be used to compute matrix-matrix multiplications, matrix inverses, and solutions to systems of linear equations, and combinations thereof. For example, the first step of solving the least squares problem $\|\mathbf{X}\boldsymbol{w}{-}\boldsymbol{y}\|$ has left the device in the state $\mathbf{V}_1{=}\mathbf{R}$ and $\mathbf{V}_2{=}\mathbf{Q}^H\boldsymbol{y}$ (see above). The solution can be completed with a transformation in the linear mode by letting $\mathbf{W}_1{=}{-}\mathbf{I}$ and $\mathbf{W}_2{=}\mathbf{0}$. Then $\mathbf{W}_2'{=}\mathbf{R}^{-1}\mathbf{Q}^H\boldsymbol{y}{=}(\mathbf{X}^H\mathbf{X})^{-1}\mathbf{X}^H\boldsymbol{y}$. By letting $\mathbf{X}$ be square, this method can clearly be used to solve arbitrary systems of linear equations with an arbitrary number of right hand sides.

To compute the arbitrary matrix-matrix product $\boldsymbol{AB}$, one can start with $\mathbf{V}_1{=}\mathbf{0}$, $\mathbf{V}_2{=}\mathbf{0}$ and input $\mathbf{W}_1{=}\mathbf{I}$, $\mathbf{W}_2{=}\boldsymbol{B}$ in a orthogonal run, leading to $\mathbf{V}_1{=}\mathbf{I}$, $\mathbf{V}_2{=}\boldsymbol{B}$. A subsequent linear run with $\mathbf{W}_1{=}\boldsymbol{A}$, $\mathbf{W}_2{=}\mathbf{0}$ will yield $\mathbf{W}_2'{=}\boldsymbol{AB}$.

This mode is useful for filters and other signal transformations such as an FFT, for example.

– In the *hyperbolic* mode, $\mathbf{\Theta}$ fulfills

$$\mathbf{\Theta}^H\mathbf{J}\mathbf{\Theta}{=}\mathbf{J} \qquad \text{with} \qquad \mathbf{J}{=}\begin{bmatrix} \mathbf{I} & \\ & -\mathbf{I} \end{bmatrix}. \tag{4}$$

This mode can be used to compute a QR downdating step, which can reverse a updating step. Similar to the updating step described above, an application of the device in hyperbolic mode will give us the upper triangular $\mathbf{R}_2$ such that $\mathbf{R}_2^H\mathbf{R}_2{=}\mathbf{X}_1^H X_1{-}\mathbf{X}_2^H\mathbf{X}_2$ when starting from $\mathbf{R}_1^H\mathbf{R}_1{=}\mathbf{X}_1^H\mathbf{X}_1$.

The hyperbolic mode can also be used to carry out one step of the *Schur Algorithm* and can thus be used to efficiently compute the QR decomposition of a matrix with a Toeplitz-derived structure (Kailath and Chun, 1994). These matrices appear in time-invariant single- and multi-user systems (Vollmer et al., 1999, 2001).

– The *set* mode is provided to initialize the array. It performs the assignments

$$\mathbf{V}_1' = diag(\mathbf{W}_1) \quad \text{and} \quad \mathbf{V}_2' = \mathbf{0}$$

where $\mathbf{W}_1 \in \mathbb{R}^{1\times n}$ is a row-vector whose elements are put on the diagonal of $\mathbf{V}_1'$.

Initializing the diagonal of $\mathbf{V}_1$ is useful to compute the *best linear estimator* in white noise, for example, which is similar to a least-squares solution and is given by the formula

$$\boldsymbol{w} = (\mathbf{X}^H\mathbf{X} + \sigma^2\mathbf{I})^{-1}\mathbf{X}^H\boldsymbol{y}.$$

Setting $\mathbf{V}_1{=}\sigma\mathbf{I}$, $\mathbf{V}_2{=}\mathbf{0}$ via a run in the set mode, and then inputting $\mathbf{W}_1{=}\mathbf{X}$, $\mathbf{W}_2{=}\boldsymbol{y}$ for a run in the orthogonal mode will lead to $\mathbf{V}_1{=}\mathbf{R}$ such that $\mathbf{R}^H\mathbf{R}{=}\sigma^2\mathbf{I}{+}\mathbf{X}^H\mathbf{X}$ and $\mathbf{V}_2{=}\mathbf{R}^{-H}\mathbf{X}^H\boldsymbol{y}$. This can be transformed into the desired solution with a final run in the linear mode: as previously, $\mathbf{W}_2{=}{-}\mathbf{I}$ and $\mathbf{W}_2{=}\mathbf{0}$ will lead to $\mathbf{W}_2'{=}\mathbf{V}_1^{-1}\mathbf{V}_2{=}\mathbf{R}^{-1}\mathbf{R}^{-H}\mathbf{X}^H\boldsymbol{y}{=}\boldsymbol{w}$.

– The *copy* mode finally can be used to retrieve $\mathbf{V}_2$ in case it is needed, such as with the Schur algorithm or when the $\mathbf{Q}$ factor of a QR decomposition is needed explicitely. It sets

$$\mathbf{V}_1' = \mathbf{V}_1, \quad \mathbf{V}_2' = \mathbf{V}_2, \quad \mathbf{W}_2' = \mathbf{V}_2.$$

4 Example: Adaptive RLS Filter

A adaptive RLS filter alternates between estimating and equalizing the transmission channel. The filter that equalizes the channel is modeled as a FIR filter and the estimation is performed while a known training sequence $\boldsymbol{y}_1$ is transmitted. The coefficients $\boldsymbol{w}$ of the equalization filter are chosen such that $\|\mathbf{X}_1\boldsymbol{w}{-}\boldsymbol{y}_1\|$ is minimized where $\mathbf{X}_1$ is the convolution matrix of the signal received during the training period. The convolution matrix $\mathbf{X}$ of a sequence $\{\ldots, x_{i-1}, x_i, x_{i+1}, \ldots\}$ has a Toeplitz structure:

$$\mathbf{X} = \begin{bmatrix} x_i & x_{i-1} & x_{i-2} & \cdots \\ x_{i+1} & x_i & x_{i-1} & \cdots \\ x_{i+2} & x_{i+1} & x_i & \cdots \\ \vdots & \vdots & \vdots & \end{bmatrix}$$

The equalization is then performed by computing $\boldsymbol{y}_2{=}\mathbf{X}_2\boldsymbol{w}$ where $\mathbf{X}_2$ is the convolution matrix of the received signal during the payload period.

The CORDIC device presented above is well suited to carry out this task. The estimation phase first sets

$\mathbf{V}_1{=}\mathbf{0}$, $\mathbf{V}_2{=}\mathbf{0}$ and then carries out the QR decomposition of $\mathbf{W}_1{=}\mathbf{X}_1$ and $\mathbf{W}_2{=}\boldsymbol{y}_1$ as explained above, giving $\mathbf{V}_1{=}\mathbf{R}$ and $\mathbf{V}_2{=}\mathbf{Q}^H\boldsymbol{y}$. The equalization phase runs a subsequent linear mode transformation with $\mathbf{W}_1{=}{-}\mathbf{X}_2$, $\mathbf{W}_2{=}\mathbf{0}$, computing $\mathbf{W}_2'{=}\mathbf{X}_2\mathbf{R}^{-1}\mathbf{Q}^H\boldsymbol{y}_1{=}\mathbf{X}_2\boldsymbol{w}{=}\boldsymbol{y}_2$.

The convolution matrices $\mathbf{X}_1$ and $\mathbf{X}_2$ are constructed implicitly by connecting the outputs of a delay-line to the inputs of the device. The device can be simply switched from the training mode to the filter mode by inputting zeros instead of the training sequence and switching the mode from *orthogonal* to *linear*.

In order to allow the filter to gradually forget the past, it is customary to change the scaling factor in of each CORDIC such that each elementary orthogonal rotation reduces the length of the involved vector by a factor of 0.97, say.

5 Conclusions

The well-known systolic QR array can be generalized to also be able to compute a wide variety of linear signal processing tasks. Implementing this generalized array with adiabatic logic offers opportunities for significant low-level optimizations that find uses for hardware resource that would otherwise sit idle. The array has been simulated with a bit-true and phase-true VHDL model by making use of a general VHDL package that allows the description of adiabatic logic on a functional level.

The result is a highly parallel, highly efficient data flow processor that can compute things like matrix/matrix products, matrix inverses, solutions to systems of linear equations, QR decompositions, least-squares solutions to overdetermined systems of equations, QR up- and down-dating steps, the core tasks of the Block-Schur algorithm, and Best Linear (Unbiased) Estimates.

References

Fischer, J., Amirante, E., Bargali-Stoffi, A., and Schmitt-Landsiedel, D.: Adiabatic circuits: converter for static CMOS signals, in: Kleinheubacher Berichte 2002, Advances in Radio Sciences, 247–251, 2003.

Haykin, S.: Adaptive Filter Theory, Prentice Hall, third edn., 1996.

Kailath, T. and Chun, J.: Generalized Displacement Structure for Block-Toeplitz, Toeplitz-Block, and Toeplitz-Derived Matrices, SIAM J. Matrix Anal. Appl, 15, 114–128, 1994.

Vetuli, A., Pascoli, S. D., and Reyneri, L. M.: Positive feedback in adiabatic logic, in: Electronics Letters, vol. 32, 1867–1869, 1996.

Vollmer, M., Haardt, M., and Götze, J.: Schur algorithms for Joint Detection in TD-CDMA based mobile radio systems, Annals of Telecommunications (special issue on multi user detection), 54, 365–378, 1999.

Vollmer, M., Haardt, M., and Götze, J.: Comparative Study of Joint-Detection Techniques for TD-CDMA Based Mobile Radio Systems, IEEE J. Select. Areas Commun., 19, 1461–1475, 2001.