A Floating-Point FPGA-Based Self-Tuning Regulator

Zoran Salič, Senior Member, IEEE, Jiaying Cao, and Sing Kiong Nguang, Senior Member, IEEE

Abstract—Recursive-least-square (RLS) algorithm is widely used in many areas with real-time implementation using digital signal processors. In this paper, the authors present a pure hardware implementation of a self-tuning regulator (STR) that uses a real-time RLS algorithm as the parameter estimator. The STR contains a controller design circuit and a controller circuit. Due to RLS computation-precision and dynamic-range requirements, the hardware implementation uses a floating-point format. The floating-point processing elements presented in this paper use parameterized design, where the number of exponents and mantissa bits can be changed as the data range and the accuracy of a specific application require. The strategies for overcoming the covariance matrix asymmetrical problem during the hardware computation and the covariance matrix resetting is introduced when the system is poorly exciting are presented. The design was verified with real-time experiments using a new testbed. The experiment results are presented.

Index Terms—Digital signal processor (DSP), estimation algorithm, field-programmable gate array (FPGA), floating-point arithmetic, floating-point format, recursive least squares (RLS), self-tuning regulator (STR).

I. INTRODUCTION

ADAPTIVE-CONTROL strategies are widely used in many areas such as motor controllers, tracking cameras, smart antennas, filters for crosstalk removal and channel estimation, etc. [1]–[6]. Most strategies have been realized using single or multiple digital signal processors (DSPs) and faced implementation problems in terms of performance and accuracy of computation. A self-tuning regulator (STR) with an online recursive-least-square (RLS) estimator is a promising adaptive-control strategy for many applications. Its digital implementation naturally lands on DSP platforms. All digital STR systems include an estimation circuit, the controller design circuit and the controller circuit modeled as shown in Fig. 1. The block “estimation circuit” estimates the process parameters online. The block labeled “controller design circuit” contains algorithms that are required to perform a design of a controller with a specified method and a few design parameters that can be chosen externally. The block labeled “controller circuit” is an implementation of the controller of which parameters are obtained from the “controller design circuit” [7].

In order to overcome the shortcomings of DSP processors in the implementation of the STR (primarily sequential processing due to program execution and insufficient computing power), an STR using the full hardware implementation on a field-programmable gate array (FPGA) with fixed-point arithmetic was proposed and implemented [8], [9]. However, the fixed-point arithmetic has been a limiting factor for most applications due to two major problems: 1) the dynamic range of computation and 2) the inflexibility to customize the hardware circuit once the features of the application are known. More discussion on this is presented in Section IV.

FPGAs have become quite common not only as prototyping but also the implementation technology for digital systems. FPGAs are a competitive alternative for high-performance DSP applications, previously dominated by general-purpose DSP processors. One example is the methodology of the scalar-based direct algorithm mapping (SBDAM) on an FPGA and is used in the implementation of a Kalman filter [10]. The approach is based on developing matrix equations into scalar form and on using only simple arithmetic operations for the implementation. A similar approach, but extended to the floating-point arithmetic, has been introduced in [11] and further developed in this paper for implementation of the STR. The new STR uses a floating-point-based parameterized data path, which can easily be customized for concrete specific application and a minimal number of arithmetic processing elements (adder, multiplier, multiplier/accumulator, and divider), which are used in a serialized manner, to implement the STR. This version of the controller is called the floating-point serial STR (FSSTR). The options of using more such units to exploit the internal parallelism of the STR are left for future studies.

Exponential forgetting is a way to discard old data exponentially. However, exponential forgetting in combination with poor excitation can result to an undesirable effect called the covariance windup [12]. To overcome this in FSSTR, a conditional covariance resetting algorithm is used, which provides desired system behaviors as will be demonstrated on a number of experiments.

In Section II, we introduce mathematical models used to describe the STR system. In this paper, the consideration is restricted on the second order systems only. Section III shows the scalar form of an RLS, which is suitable for hardware implementation. Section IV describes the floating-point format we used in our design, the FSSTR design and the design of individual processing elements (components). The STR has been developed within a flexible design and testbed platform, which is shown in Section IV. The implemented testbed offers a unique environment that fully combines hardware implementation of an FSSTR and communication with a process model simulated in the software. The experiment settings and results are shown in Section V. The discussion, conclusion, and the future study are presented in Section VI.
II. BASIC SYSTEM MODEL

In this section, we present the mathematical models of each part of the STR system from Fig. 1.

A. Process Model

It is assumed that the process is described by a single-input single-output (SISO) system

\[ A(q^{-1})y(t) = B(q^{-1})(u(t) + v(t)) \]

where \( y(t) \) is the output, \( u(t) \) is the input, \( v(t) \) is the disturbance, and \( q^{-1} \) is the delay operator. \( A(q^{-1}) \) and \( B(q^{-1}) \) are polynomials in \( q^{-1} \). It is assumed that \( A(q^{-1}) \) and \( B(q^{-1}) \) do not have any common factors. For simplicity, disturbance \( v(t) \) is assumed to be zero. Thus, process model (1) can be rewritten explicitly as

\[ y(t) = -a_1 y(t-1) - a_2 y(t-2) - \cdots - a_n y(t-n) + b_0 u(t-d_0) + \cdots + u(t-d_0-m). \]

(2)

Noting that this model is linear in the process parameters, the model is rewritten as

\[ y(t) = \varphi^T(t-1)\theta \]

(3)

where

\[ \theta^T = [a_1, a_2, \ldots, a_n, b_0, \ldots, b_m] \]

(3a)

and

\[ \varphi^T(t-1) = [-y(t-1), \ldots, -y(t-n), u(t-n), \ldots, u(t-d_0-m)]. \]

(3b)

B. RLS

The estimation circuit is the heart of an STR. The RLS algorithm with an exponential forgetting algorithm is used in this circuit. It recursively estimates the unknown process parameters for each measurement based on the minimization of the least-square error. The whole algorithm involves the following matrix computations:

\[ \epsilon(t) = y(t) - \varphi^T(t-1)\hat{\theta}(t-1) \]

(4a)

\[ K(t) = P(t-1)\varphi(t-1) \]

\[ \times \left( \lambda + \varphi^T(t-1)P(t-1)\varphi(t-1) \right)^{-1} \]

(4b)

\[ \hat{\theta}(t) = \hat{\theta}(t-1) + K(t)\epsilon(t) \]

(4c)

\[ P(t) = (I - K(t)\varphi^T(t-1)) \frac{P(t-1)}{\lambda} \]

(4d)

where \( \hat{\theta}(t) = [\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_n, \hat{b}_0, \ldots, \hat{b}_m] \) is the estimated process-parameter vector, \( \lambda \) is the forgetting factor, \( K(t) \) is the Kalman gain, \( \epsilon(t) \) is the error in predicting the signal \( y(t) \) one step ahead based on the estimate \( \hat{\theta}(t) \), and \( p(t) \) is the error covariance. In general, the estimation circuit operates at each process output sample instance as follows.

1) New data \( y(t) \) and \( \varphi(t) \) are acquired, and the prediction error is computed from the old estimated parameter (4a).
2) Equation (4b) is used to update the Kalman gain vector \( K(t) \).
3) New parameter \( \hat{\theta}(t) \) is calculated using (4c).
4) Data \( p(t) \) is updated by (4d) for the next sample.

Using this form of RLS algorithm in a number of experiments, it is found that estimated plant parameters converged to their true values and the RLS algorithm proved to be stable for hundreds of iterations of the estimator. However, it becomes unstable sometimes and is unable to properly estimate the system parameters, as shown and discussed in Section V.

C. Covariance Matrix Resetting Algorithm

The ability of an adaptive controller to track time-varying parameters is an important issue. To do so, an exponential forgetting factor is introduced in an RLS. It is based on the assumption that the least-square loss function is replaced by a loss function in which old data is discarded exponentially. The comparison of RLS with and without the forgetting factor is shown Section V-C. From the experiment, it is found that the RLS with a forgetting factor has a smaller tracking lag than...
that without a forgetting factor. Thus, the forgetting factor can be decreased to reduce the tracking lag. However, when the command signal is constant over a long period of time, the system is poorly exciting. The exponential forgetting factor will cause the covariance windup. In order to overcome this, we add the covariance $p$-matrix resetting algorithm to the RLS.

In this algorithm, there are three signals that need to be tracked: the command signal, the plant-output signal, and the estimated parameters. Tracking the command signal is to see the system excitation status (5a). The $p$-matrix is reset when estimated parameters are converged and the plant output follows the command signal. The comparison between command signal and plant-output signal is needed (5b). The most important issue is to track the convergence status of estimated parameters (5c). The condition for resetting the $p$-matrix is as follows:

$$u_c(t + 1) = u_c(t)$$

$$y(t) > u_c(t)$$

$$\theta(t + 1) = \theta(t).$$

(5a)

(5b)

(5c)

To ensure that the $p$-matrix is reset when estimated parameters converged and the plant output follows command signal, an iteration counter is set in the FSSTR to count the number of iterations for which those conditions continuously apply. If the counter reaches a certain fixed value (e.g., 10), the $p$-matrix is reset. If one of those conditions is not satisfied, the counter will be reset immediately.

D. Controller Circuit

The controller circuit implements the following equation:

$$Ru(t) = Tu_c(t) - Sy(t)$$

(6)

where $R$, $S$, and $T$ are polynomials.

E. Controller Design Circuit

The controller design circuit determines the controller that gives desired closed-loop poles. The minimum-degree pole-placement (MDPP) algorithm for the model following with zero cancellation is used in the controller design circuit 0. The controller parameters are

$$s_1 = \frac{a_{m2} - a_2}{b_0} \quad s_0 = \frac{a_{m1} - a_1}{b_0} \quad \gamma_1 = \frac{b_1}{b_0} \quad t_0 = \frac{b_m0}{b_0}$$

(7)

where $a_{m2}$, $a_{m1}$, and $b_{m0}$ are the closed-loop system parameters and $t_0$, $r_1$, $s_0$, and $s_1$ are the controller parameters.

III. SECOND-ORDER-SYSTEM SCALAR FORM OF RLS

In this paper, our discussion is restricted to a second order system. In order to implement a complex matrix equation in hardware, the equation is decomposed into its scalar form. Then, only simple arithmetic operations such as addition, subtraction, multiplication, and division are used in these scalar equations. The scalar form of the RLS algorithm is derived from (4a)–(4d).

1) Equation (4a) in the scalar form converts to

$$\varepsilon(t) = y(t) - (\varphi_1(t - 1)\theta_1(t - 1) + \varphi_2(t - 1)\theta_2(t - 1) + \varphi_3(t - 1)\theta_3(t - 1) + \varphi_4(t - 1)\theta_4(t - 1))$$

(8)

2) Equation (4b) in the scalar form converts to

$$K_i^1(t) = P_{i1}(t-1)\varphi_1(t-1) + P_{i2}(t-1)\varphi_2(t-1) + P_{i3}(t-1)\varphi_3(t-1) + P_{i4}(t-1)\varphi_4(t-1)$$

(9)

$$A = \lambda + \varphi^T(t-1)p(t-1)\varphi(t-1) = \lambda + K_i^1(t)\varphi_1(t-1) + K_i^1(t)\varphi_2(t-1) + K_i^2(t)\varphi_3(t-1) + K_i^2(t)\varphi_4(t-1)$$

(10)

$$K_i(t) = \frac{K_i^1(t)}{A}$$.  

(11)

3) Equation (4c) in the scalar form converts to

$$\hat{\theta}_i(t) = \hat{\theta}_i(t - 1) + K_i(t)\varepsilon(t)$$.  

(12)

4) Equation (4d) in the scalar form converts to

$$p_{ij}(t) = \frac{(p_{ij}(t - 1) - K_i(t)K_i^1(t))}{\lambda}$$

(13)

By counting the number of operations, it can be seen that 44 multiplications, 40 additions or subtractions, and 20 divisions are required for a fully parallel estimation block implementation. In order to reduce the number of processing elements, a decision was made to implement the circuit with a single shared copy (component) of each of the processing elements. The same approach has been used in our initial fixed-point implementation of the STR presented in [8].

IV. FSSTR

The major problem with fixed-point number representation is that it can represent a limited range of data, which causes inaccuracies in the algorithm implementation. A fixed binary or radix-point representation allows the representation of numbers with a fractional component, but the dynamic range is limited, causing problems in representing very small and very large numbers. In our early implementation of the STR, a fixed-point format with radix point 0 was used. First, the true plant parameters are chosen by implementing the STR in MATLAB. According to the computation data collected from MATLAB, the position of the radix point for the fixed-point format STR is chosen. By running a MATLAB simulation, the implementation data range was determined, which varies with different sets of true plant parameters. Obviously, this is not feasible to do in real time, as the true plant parameters are not known, making a very difficult implementation of the STR with fixed-point arithmetic.
A. Floating-Point Format

In order to avoid problems of fixed-point format in our earlier implementation of the STR [8], [9], a floating-point format, which is a combination of IEEE Standard 754 and the needs of the STR, has been specified and used. It is illustrated in Table I. Here, 8 exponent bits and 16 mantissa bits are chosen.

The floating-point numbers are in the form

\[( -1)^S \times M \times 2^E \]

where \( S \) is the sign bit, \( E \) is the exponent field as 2’s complement number with no bias used in the IEEE standard, and \( M \) is the mantissa field representing numbers in the range between 1 and 2. If the bits of the mantissa from left to right are \( M_1, M_2, M_3, \ldots \), then the value of the number is

\[( -1)^S \times (1 + M_1 \times 2^{-1} + M_2 \times 2^{-2} + M_3 \times 2^{-3} + \cdots) \times 2^E.\]

More details on arithmetic processing elements for this number format are given in Section IV-C.

B. FSSTR

The STR implemented in our case is of a serial type using only a single shared instance of all basic arithmetic elements and, hence, its name FSSTR. Our goal was an implementation that requires the minimum amount of hardware resources; however, due to sharing of processing elements, it is also the lowest speed implementation. The strategy in its design is similar to the one applied in the fixed-point case [8]. The FSSTR global architecture is shown in Fig. 2. The major difference to the previous architecture is the use of five dual-port RAM memories as the temporary storage for calculated parameters: \( p(t) \) _mem is used to store the covariance errors; \( K(t) \) _mem stores Kalman gains; \( \varphi(t) \) _mem stores delay line \( \varphi; \) \( \hat{\theta}(t) \) _mem is for estimated parameters; and \( K^1(t) \) _mem stores the intermediate results of computation. The convergence monitor circuit is used for checking the convergence level of the estimated plant parameters, and it is a part of the control unit. The data_in port is used to input data from plant \( y(t) \), initial values of \( \hat{\theta}(t) \), and initial covariance \( p(t) \). The data_out port is used to output data to plant \( u(t) \) and intermediate results of computation from memory blocks for analysis in MATLAB, as will be shown in Section V. The address is supplied from an external source (software) through the peripheral-component-interconnect (PCI) bus for fetching data from memory blocks and bringing them to the external world. The start and finish signals are the handshake signals for communication between FSSTR hardware and its external world.

C. Building Blocks

1) Floating-Point MAC: The multiply and accumulate (MAC) is the core computation unit in the FSSTR circuit. Its structure is shown in Fig. 3. It contains a floating-point multiplier, a floating-point adder/subtraction, one register, and four multiplexers. By selecting control signal of multiplexers (MUXs), this circuit can implement a single addition or multiplication and accumulation.

2) Floating-Point Adder/Subtraction: To perform basic arithmetic operations, floating-point adder/subtractor, multiplier, and divider have been designed. In floating-point arithmetic, addition and subtraction are more complex than multiplication and division. It is necessary to ensure that both operands have the same exponent before doing the addition. This requires shifting the radix point on one of the operands to achieve alignment. The floating-point adder/subtractor is shown in Fig. 4. Exponent comparator (EXCMP) and mantissa comparator (MANCMP) are used to determine which operand is larger and by how much. The difference controls mantissa multiplexer (MANMUX) to choose the exponent bits of larger number to the normalize circuit, the mantissa bits of larger number to adder/subtractor, the mantissa bits of smaller number to the shift right register, and the sign bit of bigger operand as the final sign bit output. The normalization step shifts the sum left or right and increments or decrements the exponent.

3) Floating-Point Multiplier: The floating-point multiplication is more straightforward compared with the floating-point addition. Multiplication of floating-point numbers \( a \) and \( b \) can be represented as

\[ a_M \times b_M \times 2^{(a_E + b_E)}. \]

Fig. 5 illustrates the architecture of the floating-point multiplier. The sign of the result is simply found by XORing the sign bit of multiplicands. The exponent is the sum of the two exponents. The addition of the two exponents is in 2’s complement format. The multiplication of the fraction is an integer unsigned multiplication of two mantissas. The normalization is implemented with 1-bit shift if the multiplication overflows.

4) Floating-Point Divider: As the division element has the longest propagation delay and taking the advantage of the need for division by a constant, division \( A/B \) is converted in the STR into multiplication \( A \times (1/B) \) and implemented using a look-up table (LUT) as illustrated in Fig. 6. For the purpose of normalization, 2/mantissa is stored in the LUT instead of 1/mantissa. Since the mantissa field represents numbers in the range between 1–2, 1/mantissa will be the numbers less than 1. 2/mantissa will be the numbers in the range between 1–2. For the same reason, the result of exponent field will be: \(-1 \) (–exponent of the input number).

D. Memory Blocks

There are five dual-port memory blocks in the FSSTR. These memory blocks have an 8-bit address bus, and the data width is 32 bits. \( P(t) \) _mem stores \( p(t) \), initial covariance \( p(0) \), and \( a_{m2}, a_{m3} \). The top 16 locations of memory stores \( p(t) \), and the next 16 locations is for \( p(0) \). \( \varphi(t) \) _mem stores delay line, and
the reference input $u_c$. $K(t)_\text{mem}$ stores intermediate data $K(t)$, $A$, $\varepsilon(t)$, $r_1$, and $b_{m0}$. $\hat{\theta}(t)_\text{mem}$ stores the estimated plant parameters and $\lambda$. $K(t)_\text{mem}$ stores the Kalman gain. The reason for using five dual-port memories instead of one memory block is the faster access to the memory and simultaneous access to memory locations by different processing elements.

**E. LUTs**

There are two LUTs used in the hardware circuit. One is for the divider used that is mentioned in Section IV-C4. The other LUT is used for converting the normal matrix $P$ address to fetching and computing the top triangle element of $P$ matrix only. The comparison of an ordinate $4 \times 4$ matrix and a symmetrical $4 \times 4$ matrix are shown below. There are only ten elements used because the matrix $P$ is symmetrical. It is quite easy to generate the address for the normal $4 \times 4$ matrix in the control unit. This LUT creates the symmetrical $P$ matrix address according to the ordinate address that is received from the control unit. This LUT is stored into $\hat{\theta}(t)_\text{mem}$.

<table>
<thead>
<tr>
<th>Ordinate $4 \times 4$ matrix:</th>
<th>Symmetric $4 \times 4$ matrix:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3</td>
<td>0 1 2 3</td>
</tr>
<tr>
<td>4 5 6 7</td>
<td>1 4 5 6</td>
</tr>
<tr>
<td>8 9 10 11</td>
<td>2 5 7 8</td>
</tr>
<tr>
<td>12 13 14 15</td>
<td>3 6 8 9</td>
</tr>
</tbody>
</table>
The control unit implements a finite-state machine by using two counters (c_1, c_2), a state generator, an address generator, and a control-signal generator as shown in Fig. 7. Based on the state of the counters, the state generator circuit generates six global states: compute_K(t), compute_A, compute_ε(t)_K(t), compute_θ(t), compute_p(t) and compute_controller. The control-unit finite-state machine behavior is described by the flowchart in Fig. 8. The address generator circuit generates addresses of memory blocks accessed in each individual state. The control-signal generator circuit generates the memory write signal, the data selection signal for MAC, and the multiplier and divider and enable signal for the convergence monitor circuit to check the value of product ˆθ(t). The start signal is generated by a program running on the PC and is sent via the PCI bus. The finish signal is a signal from the FSSTR to the program on the PC received via the PCI bus. The convergence signal is supplied by the convergence monitoring circuit.

G. FSSTR Prototype Implementation Platform

The FSSTR has been implemented using an FPGA PCI prototyping board. The board has a number of useful features that make it ideal for testing the hardware implementation. The FPGA device used (Altera Flex 10K200S) provides 10 160 logic elements, 98 304 memory bits, and 200 000 typical gates. This device is ideal for memory functions or integrated entire systems in a single chip [13].

The PCI core is an interface between the FSSTR hardware circuit and the host PC’s PCI bus, as shown in Fig. 9. It is used to exchange data between host PC software and the FSSTR. The ior_mailbox implements a handshaking protocol between the host software and the FSSTR hardware. The host software communicates with the FSSTR via dual-port memory blocks, enabling easy debugging and experiments using custom programs or standard tools such as MATLAB.

V. EXPERIMENTS AND RESULTS

A testbed has been established for performing various experiments with the hardware-implemented FSSTR and is shown in Fig. 10. The PCI board is the rapid prototyping platform for experiments with STRs. The host PC has three functions in the experiments. First, it provides a PCI interface with the PCI board. Also, it executes a program (written in C language) that simulates the plant process, produces the plant output for the FSSTR, and collects various intermediatedata from the FSSTR. Finally, it runs MATLAB for the purpose of data analysis and plots various graphs that illustrate the behavior of the simulated plant and the controller.

The FSSTR estimates parameters in real time and returns them to the C program and MATLAB for comparison and analysis. As the FSSTR design is parameterized, different implementation parameters are chosen such as the numbers of mantissa and exponent bits. In the experiments presented in this paper, we use a 16-bit mantissa and an 8-bit exponent. Thus, all the data collected from the experiment are in a floating-point format with a 1-bit sign, an 8-bit exponent, and a 16-bit mantissa.

In the experiments, it has to be to ensure that the p_matrix is symmetric during the iterative computation. It is difficult to achieve this if all 16 elements of the p_matrix are calculated in the computation because of the accumulation of hardware computation errors due to the limited mantissa length. In fact, only ten elements in the p_matrix are used and then updated. When p_{ij} (i > j) is needed, we use p_{ji} instead. The algorithm of the symmetric p_matrix hardware implementation is presented in Section IV-E.

The comparison of results when using symmetric and asymmetric covariance matrices is shown in Section V-A. The function of the exponential forgetting factor is discussed in
Section V-B. The experimental results of the covariance resetting algorithm are presented in Section V-C.

A. Time-Invariant Parameters

Several fixed sets of parameters are used in these experiments. The typical results are shown in Table II. The true plant parameters that are stored in the host PC are in bold. The estimated plant parameters are calculated by the FSSTR. From Table II, it can be seen that estimated plant parameters are very close to the true plant parameters.

Figs. 11 and 12 show the dynamic behavior of both asymmetric and symmetric covariance matrices for the first set of parameters in Table II, respectively. The command signal is a square wave of unit amplitude and with period of 50 samples. Thus, it is assumed that the system is persistently excited. Both matrices find their true values, but the FSSTR with an asymmetric covariance matrix shows wrong behaviors at 900 iterations in Fig. 11(a). The error is oscillated widely as shown in Fig. 11(c). The comparison between the plant output and command-signal input $u_c$ is shown in Fig. 11(b). The plant output does not follow the command signal all the way through. In Fig. 11(a), it can be seen that the estimated parameter $a_0$ tracks the true parameter very well at the beginning. This is because we initialize the covariance $p$ matrix as symmetric before the FSSTR start implementation. After hundreds of
TABLE II
PARAMETER ESTIMATION—EXPERIMENTAL DATA

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>$a_0$</th>
<th>$a_1$</th>
<th>$b_1$</th>
<th>$b_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>True plant parameter</td>
<td>-1.6065</td>
<td>0.6065</td>
<td>-0.1065</td>
<td>0.0902</td>
</tr>
<tr>
<td>Estimated plant parameter</td>
<td>-1.60553</td>
<td>0.60580</td>
<td>-0.10645</td>
<td>0.09017</td>
</tr>
<tr>
<td>True plant parameter</td>
<td>2.56</td>
<td>1.56</td>
<td>0.67</td>
<td>0.1065</td>
</tr>
<tr>
<td>Estimated plant parameter</td>
<td>2.5578</td>
<td>1.5580</td>
<td>0.6689</td>
<td>0.1062</td>
</tr>
<tr>
<td>True plant parameter</td>
<td>-5.78</td>
<td>4.78</td>
<td>1.1065</td>
<td>0.0902</td>
</tr>
<tr>
<td>Estimated plant parameter</td>
<td>5.7758</td>
<td>4.7760</td>
<td>1.1054</td>
<td>0.09005</td>
</tr>
</tbody>
</table>

Fig. 11. Tracking time-invariant parameters with an asymmetric covariance $p$-matrix algorithm. (a) Parameter $a_0$. (b) Plant output. (c) Error refers to (4a).

Fig. 12. Tracking time-invariant parameters with a symmetric covariance $p$-matrix algorithm. (a) Parameter $a_0$. (b) Plant output. (c) Error refers to (4a).

iterations, the covariance $p$-matrix is not symmetric at all due to the accumulation of hardware computational errors.

Good results are obtained when a symmetric $p$-matrix algorithm is used in the FSSTR. This is shown in Fig. 12. From these figures, it is obvious that the FSSTR without a symmetric $p$-matrix algorithm is unstable.

B. Time-Varying Parameters

In this section, an experiment when true-parameter values change instantaneously by 5% at 300th and 600th FSSTR iterations is presented. The same command input signal is used. That is, a square wave of unit amplitude and a period of 50 samples.

Also, the system is exciting. The FSSTR with the asymmetric covariance $p$-matrix cannot track the change of true parameters; the plant output does not follow the command signal, and the error is high. This is shown in Fig. 13. In Fig. 13(a), the estimated parameters track their first true value, but they cannot track second and the third values. Due to the accumulation of hardware computation errors, covariance $p$-matrix is not symmetric. This is the reason why the error is small in the beginning (iterations 0–250) as shown in Fig. 13(c).

In Fig. 14(a), the estimated parameter $a_0$ tracks its true value, but there is a track lag between the true and estimated parameter when the true parameter is changing. This situation will be discussed in the next experiment. The plant
output follows the command signal in Fig. 14(b). The error is small except at the 300th and the 600th iterations shown Fig. 14(c), at the points where the true parameter is changing its value.

C. Exponential Forgetting Factor

In the previous two sections, it is shown that the asymmetric covariance \( p \_ \) matrix will yield an unstable system behavior. In the rest of the experiment, the symmetric covariance \( p \_ \) matrix algorithm is used in the FSSTR. The tracking lag between estimated- and true-parameter values appears to be due to the use of the exponential forgetting factor in FSSTR. For the experiment shown in Fig. 14, the exponential forgetting factor was set to 0.9.

In the experiment shown in this section, the same command signal is used as in the experiment in Section V-B. However, the exponential forgetting factor is 1. The estimated parameter \( a_0 \) tends to track the true-parameter value. It discards the old data too slow to track the changes of the true parameter in
In Fig. 16, the exponential forgetting factor is 0.95. The estimated parameter $a_0$ tracks the change of the true parameter. However, there is a tracking lag between estimated and true parameters in Fig. 16(a). The tracking lag is bigger than the one shown in Fig. 14.

In Fig. 17, the exponential forgetting factor is 0.7. The estimated parameter $a_0$ tracks the change of the true parameter very well. The tracking lag is reduced significantly as shown in Fig. 17(a). The error is small except at the 300th and the 600th iterations when the true parameter is instantaneously changing its value.

From the experiments, it can be seen that the function of the exponential forgetting factor is to discard old data exponentially.

### D. Covariance $p$-Matrix Resetting

In the previous experiment, the command signal was set as a square wave with a unit amplitude and a period of 50 samples. It is also assumed that the system is well exciting and the major concern is the effect of the exponential forgetting factor.

In this section, the ability of the FFSTR to cope with a poorly excited system when the command signal is constant over long period of time is discussed. Exponential forgetting is a way to discard old data exponentially. However, the exponential forgetting in combination with a poor excitation can be a cause of an undesirable effect called covariance windup. In the extreme case, there is no excitation at all, that is, $\varphi = 0$. The equations for the estimate then become

$$\theta(t + 1) = \theta(t) \quad P(t + 1) = \frac{1}{\lambda} P(t).$$

The estimate will remain constant, and the $p$-matrix will grow exponentially if $\lambda < 1$ [7]. The covariance $p$-matrix resetting is needed when the excitation is poor. Nevertheless, a unit square wave is used for command signal, but the command-signal period is 400 samples. The comparison between the FSSTRs with and without covariance $p$-matrix resetting is shown in following figures. Fig. 18 shows the system without covariance $p$-matrix resetting. The element $p_{00}$ of the $p$-matrix grows approximately exponentially during the periods when the command signal is constant and drops quickly when the command signal is changing as in Fig. 18(c). The other elements of $p$-matrix behave similarly. Covariance $p_{00}$ reaches to $10^7$. The estimated parameter $a_0$ will change significantly as shown in Fig. 18(a). The FSSTR with covariance $p$-matrix resetting algorithm is stable. The change region of covariance $p_{00}$ is 0–120. This is shown in Fig. 19. The parameter $a_0$ tracks its true value.

From the experiment, it can be seen that covariance resetting reduces the $p$-matrix exponential windup. Thus, exponential
VI. CONCLUSION AND FURTHER STUDY

A fully hardware implementation of the FSSTR, which uses covariance $p$-matrix resetting and exponential forgetting for real-time estimation of plant parameters, is presented in this paper. The FSSTR with the covariance resetting algorithm provides stability of computation of plant parameters and controller, which was not guaranteed when the system is poorly excited if we use the common RLS algorithm. The FSSTR is implemented in a single FPGA device with the sharing of basic arithmetic processing elements performing operations in a floating-point format. The implementation uses relatively moderate FPGA resources: 37% of resources of a low-capacity Altera FLEX10K200 device. The design uses advantages offered by this type of FPGA device, which provides distributed small memory blocks that can operate as dual-port memories to store intermediate results of computation. To perform the system parameter estimation and control action, the circuit requires 60 clock cycles. Using a very moderate clock with a 100-ns cycle time, a single control step is performed within 6 ms, allowing a sampling frequency of plant output at 166 000 samples/s. A testbed has been built to provide efficient experiments with the real implementation of the FSSTR with various plants that are simulated by a program running on a PC. This testbed environment enables further developments of the concept and its extensions in various directions. A future study is to explore the implementation of higher order system control using self-tuning controllers and the utilization of parallelisms in the algorithm to achieve a higher performance FSSTR aimed at a higher plant-output sampling frequency.

REFERENCES


Zoran Salcic (S’75–M’76–SM’98) received the B.E., M.E., and Ph.D. degrees from the University of Sarajevo, Sarajevo, Bosnia and Herzegovina, in 1972, 1974, and 1976, respectively, all in electrical engineering.

He is a Professor and the Chair in Computer Systems Engineering, Department of Electrical and Computer Engineering, The University of Auckland, Auckland, New Zealand. His research interests include complex digital-system architectures and design, prototyping and customization of digital systems, and new architectures and formal models for embedded systems for combined data- and control-driven applications. He is the author of more than 150 papers in journals and refereed conferences, seven books, and numerous technical reports. He led or took part in several dozens of research projects and complex digital and computer-based system designs in over 30 years. He established and leads the Embedded Systems Research Group, The University of Auckland.

Jiaying Cao received the M.E. degree in electrical and electronic engineering from The University of Auckland, Auckland, New Zealand, in 1999.

She has worked in the electronics industry for more than ten years. Her major areas of interest are the implementation of complex algorithms in hardware, particularly, field-programmable gate arrays, complex digital systems and design techniques, rapid prototyping, and self-tuning controllers.

Sing Kiong Nguang (M’97–SM’00) received the B.E. degree (with first class honors) and the Ph.D. degree in 1992 and 1995, respectively, from the Department of Electrical and Computer Engineering, University of Newcastle, Callaghan, Australia.

He is currently a Senior Lecturer in the Department of Electrical and Computer Engineering, The University of Auckland, Auckland, New Zealand.

He is the author of more than 60 journal papers and over 40 conference papers/presentations on nonlinear control design, nonlinear H-infinity control systems, nonlinear time-delay systems, nonlinear sampled-data systems, biomedical systems modeling, fuzzy modeling and control, biological systems modeling and control, and food and bioproduct processing.

Dr. Nguang is currently the Associate Editor of the IEEE Control System Society Conference Editorial Board.