

FPGA Implementation of Neural Network-Based AGPC for Nonlinear F-16 Aircraft Auto-pilot Control: Part 1 – Modeling, Synthesis, Verification and FPGA-in-Loop Co-Sim

Vincent Andrew Akpan^{1,*}, Dimitrios Chasapis², George Dimitriou Hassapis³

¹Department of Biomedical Technology, The Federal University of Technology, Akure, Nigeria

²Barcelona Supercomputing Center, Group Sonar, Barcelona, Spain

³Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Thessaloniki, Greece

Email address:

vaakpan@futa.edu.ng (Vincent Andrew Akpan), dchasapi@bsc.es (Dimitrios Chasapis), ghass@ece.auth.gr (George Dimitriou Hassapis)

*Corresponding author

To cite this article:

Vincent Andrew Akpan, Dimitrios Chasapis, George Dimitriou Hassapis. FPGA Implementation of Neural Network-Based AGPC for Nonlinear F-16 Aircraft Auto-pilot Control: Part 1 – Modeling, Synthesis, Verification and FPGA-in-Loop Co-Sim. *American Journal of Embedded Systems and Applications*. Vol. 9, No. 1, 2022, pp. 6-36. doi: 10.11648/j.ajes.20220901.13

Received: April 24, 2022; **Accepted:** June 9, 2022; **Published:** September 5, 2022

Abstract: Model predictive control (MPC) is an advanced receding horizon control strategy, for difficult multivariable control systems, that leads to constrained optimization problems which are solved online at each sampling time interval, and takes full advantage of the computational power available in modern control computer hardware for hard real-time constraint systems with short sampling time. However, nonlinear MPC (NMPC) attracts additional computational overload to satisfy nonlinear systems with hard real-time constraint and relatively short sampling time. In order to deploy NMPC for the control of nonlinear systems with hard real-time constraint and relatively short sampling time, a new model-based design (MBD) approach for the implementation of nonlinear MPC, called adaptive general predictive control (AGPC), on field programmable gate array (FPGA) for the auto-pilot control of a nonlinear F-16 aircraft is presented in this paper. The new MBD approach consists of four parts: (i) In the model identification part, the nonlinear F-16 aircraft model is approximated by a neural network autoregressive moving average with exogenous inputs (NNARMAX) model which is trained by an adaptive recursive least squares (ARLS) algorithm; (ii) In the adaptive control part, the nonlinear F-16 aircraft is controlled by a constrained neural network-based adaptive generalized predictive control (AGPC) algorithm; (iii) The third part is the online closed-loop NNARMAX model identification and AGPC control of the nonlinear F-16 aircraft; and (iv) The modeling, synthesis, verification and FPGA-in-the-loop hardware co-simulation (HW Co-Sim) of the online closed-loop NNARMAX model identification and AGPC control of the nonlinear F-16 aircraft. The training and validation data for the neural network model identification are obtained from the open-loop simulation of first-principle nonlinear F-16 aircraft model. The online closed-loop NNARMAX model identification and AGPC control of the nonlinear F-16 aircraft demonstrates the efficiency of the ARLS and the AGPC algorithms in tracking the desired reference trajectories of the nonlinear F-16 aircraft. The FPGA-in-the-loop hardware co-simulation of the online closed-loop NNARMAX model identification and AGPC control of the nonlinear F-16 aircraft shows significant reduction in the computation time between the floating-point MATLAB AGPC and fixed-point C++ AGPC algorithms. Hence, the effort in this work has been directed towards reducing the computation time of the AGPC algorithm at each sampling time instant through modeling, synthesizing and mapping the AGPC algorithm to Virtex-5 FX70T ML507 FPGA embedded system development board via FPGA-in-the-loop hardware co-simulation verification which has been successfully achieved and validated.

Keywords: Adaptive Generalized Predictive Control, Adaptive Recursive Least Squares, Auto-pilot Control, Hardware Co-simulation, Nonlinear Model-Based Design, NMPC, Neural Networks, Nonlinear F-16 Aircraft

1. Introduction

The nonlinear F-16 fighter aircraft dynamics as well as aircraft dynamics are in general nonlinear, time-varying and uncertain [1–6]. Traditionally, the nonlinear F-16 aircraft auto-pilot control system are designed by using the mathematical model of the aircraft linearized at different flight conditions such as steady wings-level, roll, pull-up or turning flights [1, 2, 4–7]. Recent approaches to F-16 aircraft modeling and control have been directed towards the use of neural network (NN) technologies with some successful results [4, 5, 7]. An intelligent adaptive control scheme based on proportional-integral-derivative (PID) controller combined with neural networks (NN) for the F-16 model estimation has been reported [7, 8]; while a more advanced control scheme based on model predictive control (MPC) technology using an adaptive generalized predictive controller (AGPC) has also been reported [4, 5]. It is observed that the F-16 aircraft auto-pilot control performances reported in the latter outperforms that reported in the former. In both papers, the F-16 aircraft auto-pilot control system compared are the roll, pitch and yaw rates control by manipulating the aileron, elevator and rudder deflections respectively as the control inputs. While the real-time constraints satisfaction of 0.5 seconds which is the sampling time for the F-16 aircraft was not satisfied [4, 7], the average computation time for the neural model identification and the adaptive AGPC F-16 auto-pilot control was reported to be approximately 6.1048 seconds [4]. Thus, it was concluded that the current multi-core computer technologies are not efficient for implementing such MPC algorithms [4, 5].

MPC is a well-established advanced control technology based on the optimization of an objective function within a specified horizon and has been recognized as the winning alternative for constrained multivariable control of industrial systems [9–11]. The main disadvantage of MPC algorithms is the computational complexity since it is based on the optimization of an objective function and this has limited the use of MPC schemes to systems with sampling time long enough to complete and update the MPC optimization problem [5]. Therefore, a more efficient computing platform is required for adapting the MPC algorithm for the nonlinear F-16 aircraft auto-pilot control.

However, researchers in the MPC community have resort to the use of field programmable gate array (FPGA) as an alternative platform for implementing MPC algorithms and attempts by several authors towards the implementation of MPC algorithms on FPGAs have been reported [12–20]. For example, reports have emerged for the implementation of MPC algorithms on FPGAs for: (i) Cessna Citation 500 aircraft pitch rate and altitude control with RC203 board incorporating the Xilinx Virtex-2 FPGA chip [12]; (ii) F-16 aircraft roll rate, pitch rate and altitude control using Xilinx Spartan 3A DSP FPGA [13, 21]; (iii) FPGA implementation of a satellite altitude control using variable structure control [22]; (iv) FPGA based longitudinal and lateral controller

implementation for a small unmanned aerial vehicle (UAV); (v) Active suspension system control with a CAP9 emulation kit which utilizes a Virtex-2 FPGA chip [18]; (vi) Antenna rotation control with Xilinx Virtex-4 FPGA board [19]. Except in the work of Vouzis [19], where an assembly-like code generator called VITO is used; all other listed papers [12–23] on FPGA implementation uses a C-like language called Handel-C obtained with the DK Design Suite from Mentor Graphics [12]. In addition to these papers, advances in FPGA implementation of MPC algorithms with several references can be found in [5, 19].

While the performances of MPC algorithms are closely related with the accuracy of the process models, all the MPC algorithms implementation reported in these papers [12–23] are based solely on first principle mathematical models which may degrade the MPC control performance when the process is operated outside its operating region [24]. Furthermore, as reported in these papers, the processor on the host computer runs portions of the MPC algorithms while the matrix computations are off-loaded to the FPGA devices such as the look-up tables (LUT), flip-flops, etc. Rather than using the host computer processor, several advantages may be derived by utilizing the embedded processor(s) available on the FPGA device [5, 25–29]. In this way, the MPC algorithm formulated in these papers could have been incorporated as a co-processor to meet real-time.

In this paper, a new model-based design approach to the FPGA implementation of a nonlinear neural network-based MPC algorithm, called the adaptive generalized predictive control (AGPC) algorithm, is presented. The neural model of the nonlinear F-16 fighter aircraft is identified and AGPC algorithm is used to obtain the control actions on the controlled variables using an Intel®Core™2 CPU running at 1.86 GHz. Using the Xilinx design suite of tools, the AGPC algorithm is synthesized, validated and FPGA-in-the-loop hardware co-simulation is performed using Virtex®-5 FX70T ML507 FPGA board and the Intel®Core™2 computer. FPGA-in-the-loop hardware co-simulation is an important aspect of FPGA design because it shows how the intended algorithm will perform when deployed for the actual target application [26, 29].

The rest of the paper is organized as follows. The dynamics and the auto-pilot control problem of the nonlinear F-16 aircraft are presented in Section 2 while the neural network model identification scheme and the neural network-based AGPC control scheme are presented in Section 3. The closed-loop implementation and simulation results for the model identification and AGPC algorithms for the nonlinear F-16 aircraft are presented in Section 4. In Section 5, the modeling, synthesis, validation and FPGA-in-the-loop hardware co-simulation of the model identification and neural network-based AGPC algorithmic model are presented. Discussions and remarks on the simulation results are given in Section 6. Section 7 concludes the paper and highlights the major contribution of the paper with some recommendation for further work.

2. The Nonlinear F-16 Aircraft: Dynamics, Auto-pilot Control Problem and Simulation

The nonlinear F-16 aircraft dynamics is, in general nonlinear, time-varying and uncertain. Traditionally, aircraft flight control systems are designed by using the mathematical model of the aircraft linearized at various flight conditions. The aircraft motion variables are sensed and fed into the aircraft control system which adjusts the surface actuators via some feedback gains. The adjustment process is called gain scheduling. Since controller designs are performed off-line using a limited number of linear and time-invariant models, extensive gain scheduling computation is required. While this approach may handle mild nonlinearities, it is not suitable for highly nonlinear problems associated with the aircraft. The gain scheduling approach may produce a control law that is applicable around the current design operating points but not globally. Thus, as aircrafts become more complex, traditional design methods have not yielded acceptable performance. To overcome these problems, nonlinear control techniques such as feedback linearization have been studied as alternatives to gain scheduling [30–32]. The use of these techniques is difficult because they depend heavily on accurate knowledge of the aircraft dynamics. Thus, a totally different approach to the nonlinear F-16 aircraft flight control is presented in this work based on the use of neural network-based nonlinear modeling and adaptive control techniques.

2.1. Formulation of the Nonlinear F-16 Aircraft Control Problem

The F-16 aircraft can be controlled by manipulating the deflections of the aileron, elevator and the rudder surfaces as well as the thrust illustrated in Figure 1 while the definition of

the angle of attack α and the angle of side slip β together with respect to the orientation of the navigation frame are shown in Figure 2. A positive aileron, elevator or rudder deflections gives a decrease in roll rate p , pitch rate q or yaw rate r respectively while a positive thrust t causes an increase in acceleration along the longitudinal body axis. The control of the nonlinear F-16 aircraft is discussed here with respect to the right control surfaces shown in Figure 1 and Figure 2.

The main objective here for the nonlinear F-16 aircraft is on neural network model identification and adaptive model-based control of the auto-pilot control system based on the orientations of the three right control surfaces; that is, the aileron, elevator and the rudder deflection control by manipulating the roll, pitch and the yaw actuator rates respectively as well as the throttle command for controlling the thrust according to the desired flight route. The desired routes of the proposed flight are illustrated in Figure 3 and are described below as follows:

- 1) It is assumed that the aircraft is ready for take off with the three control surfaces at the 0° with the thrust at full power.
- 2) The aircraft begins its straight motion by first ascending. This requires that the elevator deflects completely downwards while the aileron and the rudder still remain at 0° .
- 3) While still ascending, the aircraft banks completely to the left. This requires the aileron deflects upwards and rudder to deflect to the right.
- 4) Next, the aircraft remains on a steady flight and the control surfaces are maintained at approximately 0° while the thrust at approximately its average value to keep the aircraft in motion.
- 5) Next, the aircraft descends while banking to the right. Thus, requiring the elevator to deflect upwards approaching its maximum value whereas the aileron deflects downwards and the rudder deflects to the left.

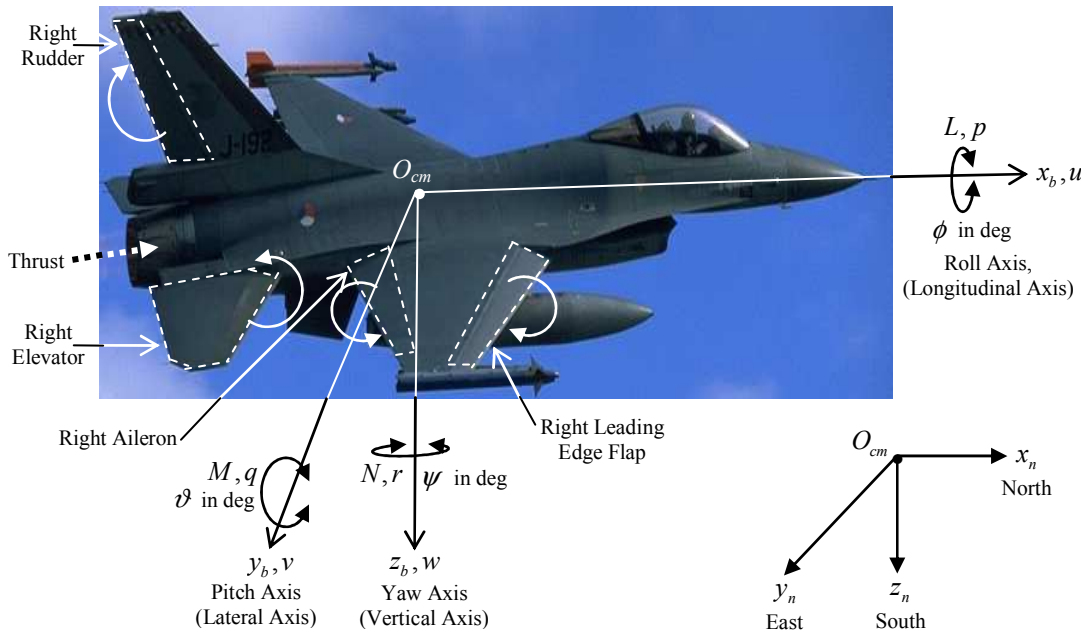


Figure 1. The nonlinear F-16 aircraft surfaces for the control of the thrust, roll rate (p), pitch rate (q), yaw rate (r), (x_b, y_b, z_b) are the body axes, (u, v, w) are the velocities along the body axes, (L) is the rolling moment, (M) is the pitching moment, (N) is the yawing moment, (x_n, y_n, z_n) is the navigation frame, O_{cm} is the center of mass, (ϕ, θ, ψ) are the Euler angles for aileron, elevator and rudder deflections respectively.

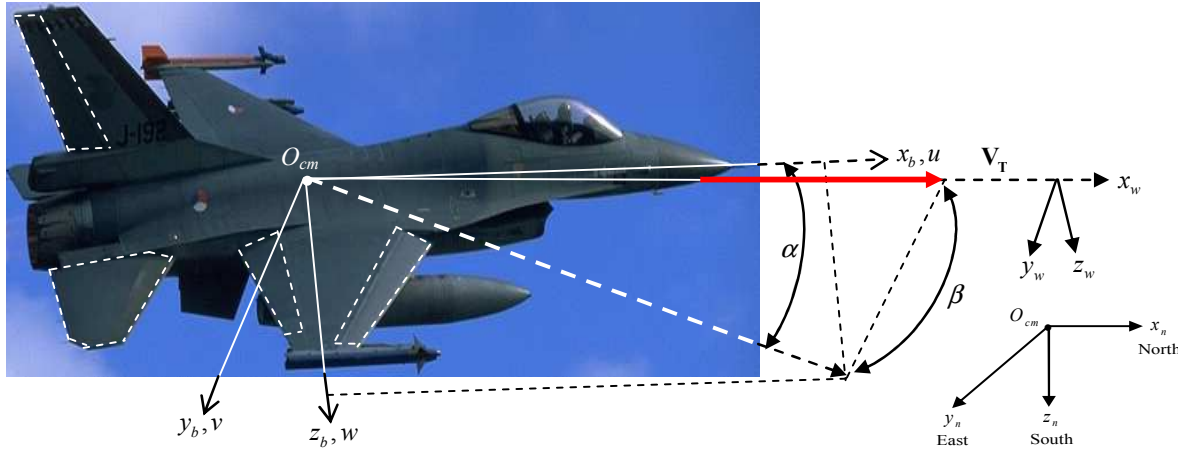


Figure 2. Definition of the angle of attack, α ($\alpha > 0$) and sideslip, β ($\beta > 0$). x_n , y_n , z_n are the North, East and South orientation of the navigation frame.

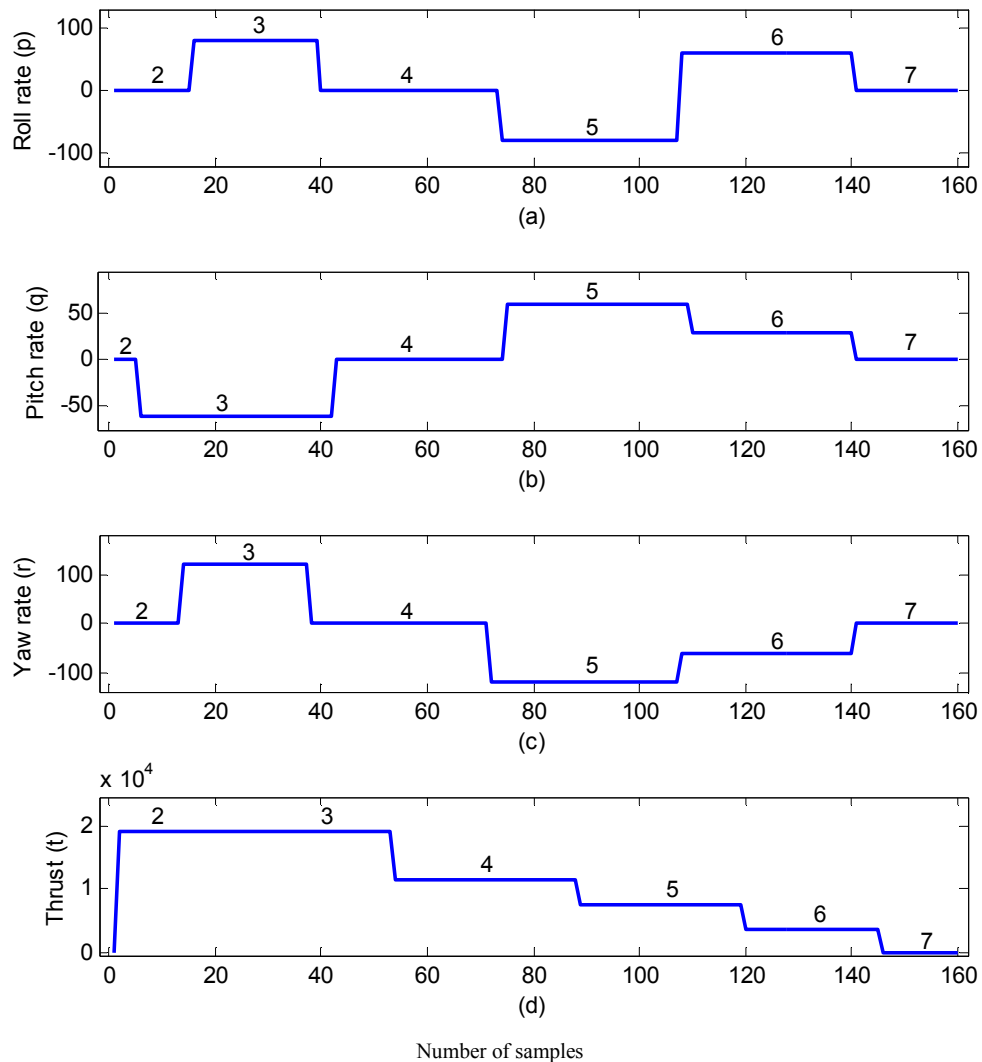


Figure 3. The desired reference trajectories for the roll rate (p), pitch rate (q) and the yaw rate (r) as well as the thrust (t) for the nonlinear F-16 aircraft.

6) Next, the aircraft makes a slight rolling action to the left and the aileron deflects upwards. At this point the elevator deflection is slightly downward approaching its zero axis (i.e. the neutral position) while the rudder

deflection is slightly increased and approaching the zero axis. The zero axis (i.e. the neutral position) is used to describe the orientation when the aircraft is at rest on the earth surface where deflections are 0° with

respect to the center of mass. This behaviour of the elevator and the rudder indicates that the aircraft descends slightly with a slight turning to the left.

- 7) Finally, the aircraft descends and all the deflection surfaces approach to 0° as the aircraft is landing and comes to rest. The thrust is also reduced to its minimum value of zero as the aircraft comes to rest.

It is important to note that while the aircraft can roll completely about the longitudinal axis using the ailerons, the turning of the aircraft about vertical axis is accomplished not only by the rudders but also in conjunction with the ailerons [33, 34]. The desired reference trajectories for the six intermediate routes discussed in step 2) to step 7) above are illustrated in the figure of Figure 3 together with the desired thrust variations.

In state space form, the motion of the nonlinear F-16 aircraft can be represented by the following nonlinear vector differential equation [35, 36]:

$$\dot{X} = g(U, X, C) \quad (1)$$

where g is a nonlinear function, U , X and C are the input vector, state vector, and the aerodynamic coefficients respectively. The thirteen states that are used to describe the rigid-body motion of the nonlinear F-16 aircraft over a flat Earth is given by:

$$X = [p_N \ p_E \ h \ \phi \ \vartheta \ \psi \ V_T \ \alpha \ \beta \ p \ q \ r \ \delta_{lef}]^T \quad (2)$$

where p_N , p_E , h , ϕ , ϑ , ψ , V_T , α , β , p , q , r and δ_{LEF} are the north position, east position, altitude, roll angle, pitch angle, yaw angle, velocity, angle of attack, angle of side-slip, roll rate, pitch rate, yaw rate and the deflection of the leading edge flap respectively. A complete model description and implementation of the nonlinear F-16 aircraft dynamics can be found in [5]. The dynamics of the nonlinear F-16 aircraft are modeled and implemented using Simulink and MATLAB C programs [37].

The aerodynamic coefficients are functions of some of the states, namely: the damping coefficients are functions of the angle of attack α ; the body-axis aerodynamic force coefficients are functions of α , β , δ_a , δ_e and δ_r ; the moment coefficients are functions of α , β and δ_e ; the coefficients of the rolling moment due to the ailerons and the rudder deflections as well as the coefficients of the yawing moment due to the ailerons and the rudder deflections are functions of α and β [2]. The input vector to the nonlinear F-16 aircraft model is

$$U(k) = [\delta_a \ \delta_e \ \delta_r \ \delta_t]^T \quad (3)$$

where δ_a , δ_e , δ_r and δ_t are the aileron deflection, elevator deflection, rudder deflection and throttle setting respectively. The eighteen outputs of the nonlinear F-16 model are: the twelve derivatives of the state variables of p_N , p_E , h , ϕ , ϑ , ψ , V_T , α , β , p , q , r ; three

normalized acceleration coordinates A_{nx} , A_{ny} , A_{nz} ; Mach number M ; dynamic pressure ($qbar$); and static pressure (p_s). The δ_{LEF} is a function of α , $qbar$ and p_s which allows the nonlinear F-16 aircraft to fly at higher angle of attack [6]. The definition of the angle of attack α with respect to the orientation of the navigation frame is shown in Figure 2. The inclusion or not of the deflection of the leading edge flap δ_{LEF} in nonlinear F-16 aircraft simulation leads to two types of models for the nonlinear F-16 aircraft, namely: (i) low fidelity and (ii) high fidelity models [1]. The low fidelity model excludes the effects of the δ_{LEF} [2] whereas the high fidelity model considers the full effects of the δ_{LEF} [6]. The differences between these two types of models are based on the data used to model the aircraft [2, 5, 6].

2.2. Simulation of the Nonlinear F-16 Aircraft for Training Data Acquisition

The F-16 turbofan engine model consists of a first-order dynamic model, a throttle command shaping function and tables of nonlinear thrust functions of the operating power level, altitude, and Mach number at different altitudes h [2]. There is one table for each of the power level called the idle, military, and the maximum [2, 6]. In the thrust lookup tables, Mach number variation is from 0 to 1 in step of 0.2 [6]. Altitude variation is from 0 to 50,000 ft in steps of 10,000 ft [6].

The aerodynamic data for the high fidelity model used in this work are based on the wind-tunnel tests on a scaled model of the nonlinear F-16 aircraft provided in [1, 6]. The values of the dimensionless aerodynamic coefficients are presented in multi-dimensional lookup tables associated with linear interpolation algorithms [6]. The aerodynamic data are referenced to the nominal position of the center of gravity $X_{CG} = 0.35$. The angle of attack ranges from -10° to 45° in steps of 5° , the sideslip angle ranges from -30° to 30° in steps of 5° , and the speed upper limit is 0.6 Mach. The limits of the actuators used to control the nonlinear F-16 aircraft are:

- (i) the aileron, elevator, rudder and the leading edge flap deflections (δ_{LEF}) are $\pm 21.5^\circ$, $\pm 25^\circ$, $\pm 30^\circ$ and 0 to 25° respectively;
- (ii) the thrust of the turbofan engine are from 1,000 to 19,000 *lbs*;
- (iii) the throttle setting rate is $\pm 10,000$ *lbs*; and
- (iv) the roll, pitch, yaw and the δ_{LEF} actuator rate are $\pm 80^\circ$, $\pm 60^\circ$, $\pm 120^\circ$ and $\pm 25^\circ$ respectively.

In order to model the nonlinear F-16 aircraft that could fly at higher angle of attack, the NASA data for the high fidelity nonlinear F-16 aircraft model [6] is used in this simulation study. The National Agency for Space Administration (NASA) data include a model of the F-16 afterburning turbofan engine, in which the thrust response is modeled with a first-order lag implemented using Simulink model shown in Figure 4. The lag time constant is a function of the actual engine power level and the throttle setting (or command).

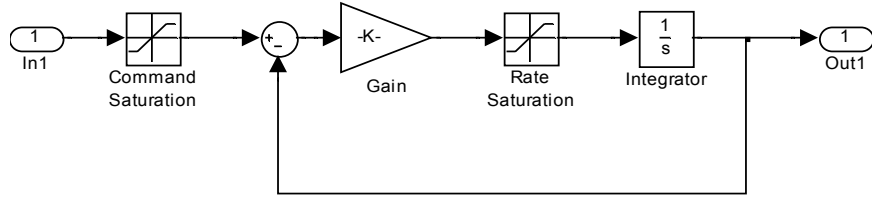


Figure 4. The Simulink actuator model for the aileron, elevator, rudder, thrust and the leading edge flap for the F-16 aircraft.

Thus, given the initial control inputs as the aileron deflection δ_a , elevator deflection δ_e , rudder deflection δ_r , throttle setting δ_t together with their respective disturbances \tilde{d}_a , \tilde{d}_e , \tilde{d}_r and \tilde{d}_t as well as arbitrary values for the states defined in Equation (2) excluding the last term δ_{LEF} ; the nonlinear F-16 aircraft can be trimmed to initial to steady wings-level, roll, pull-up or turning flight condition. After the aircraft has taken one discrete time step, the twelve input states are found from integrating the twelve outputs state derivatives given by (1) together with three normalized accelerations A_{nx} , A_{ny} , A_{nz} ; Mach number M ; dynamic pressure ($qbar$); static pressure (p_s) and the deflection of the leading edge flap δ_{LEF} . Thus, for different values of the control inputs and the type of flight condition, the nonlinear F-16 aircraft is simulated for a specified time. The number of times the trimming algorithm is called is set to 4 throughout in this work. This number of trimming gave the same steady state results in most cases.

The nonlinear F-16 aircraft is modeled using Simulink from

The MathWorks [37] and is shown in Figure 5 while the functional Simulink model of Figure 5 is shown in Figure 6. The Simulink model for the nonlinear F-16 aircraft cockpit, that is, the pilot/control input, is shown in Figure 6 while the Simulink model for the leading edge flap (LEF) is shown in Figure 7.

Using the above information, the nonlinear F-16 aircraft model parameters are varied between their minimum and maximum values over 100 different values with their respective steps as shown in Table 1 for the purpose of obtaining sufficient data for neural network training and validation. Next, the Simulink model of the nonlinear F-16 aircraft shown in Figure 5 is simulated in open-loop for four different flight conditions. These conditions are: steady wings-level flight, rolling flight, pull up/down flight, and turning flight. At each flight simulation 1,100 data were obtained, each for 25 steady wings-level, roll, pull-up/down and turning flight conditions to obtain 4,400 data samples. The last 100 data from each flight simulation making up 400 data samples (10% of 4,400) were reserved for network validation while the remaining 4,000 (90% of 4,400) were used for network training.

Table 1. Nonlinear F-16 aircraft model simulation parameters for neural network training and validation data acquisition.

S/N	Parameters	Minimum Value	Maximum Value	Steps
1	North and East position, P_N and P_E (ft)	0	180	1.8081
2	Altitude, h (ft)	5,000	50,000	454.5455
3	Aileron deflection, ϕ (deg)	-80	+80	1.2121
4	Elevator deflection, ϑ (deg)	-60	+60	1.6162
5	Rudder deflection, ψ (deg)	-120	+120	2.4242
6	Disturbances on deflections, \tilde{d} (deg)	1	12	0.1111
7	Thrust, t (lb)	10,000	19,000	90.9091
8	Velocity, V_T (ft/s)	300	900	6.0606
9	Roll rate, p (rad/s)	-21.5	+21.5	0.4343
10	Pitch rate, q (rad/s)	-25	+25	0.5051
11	Yaw rate, r (rad/s)	-30	+30	0.6061
12	Angle of attack, α (rad/s)	-10	+45	1.0101
13	Angle of side slip, β (deg)	-10	+45	1.0101
14	Throttle settings (lb/s)	1,000	10,000	90.9091

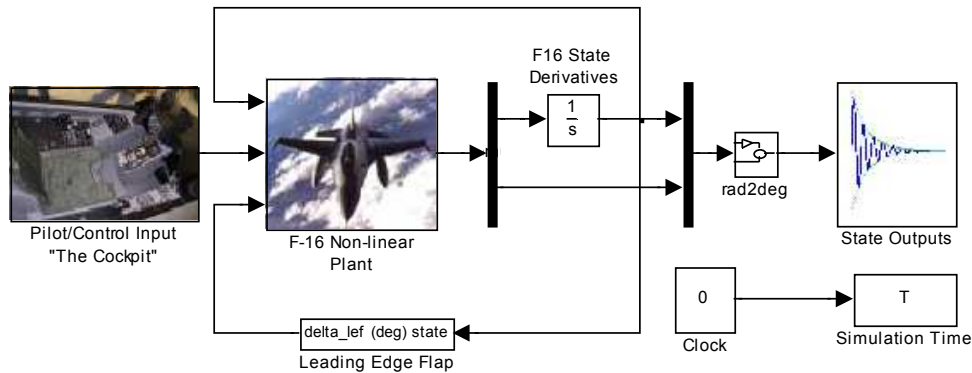


Figure 5. The schematic of the Simulink® model of the nonlinear F-16 aircraft of Figure 1.

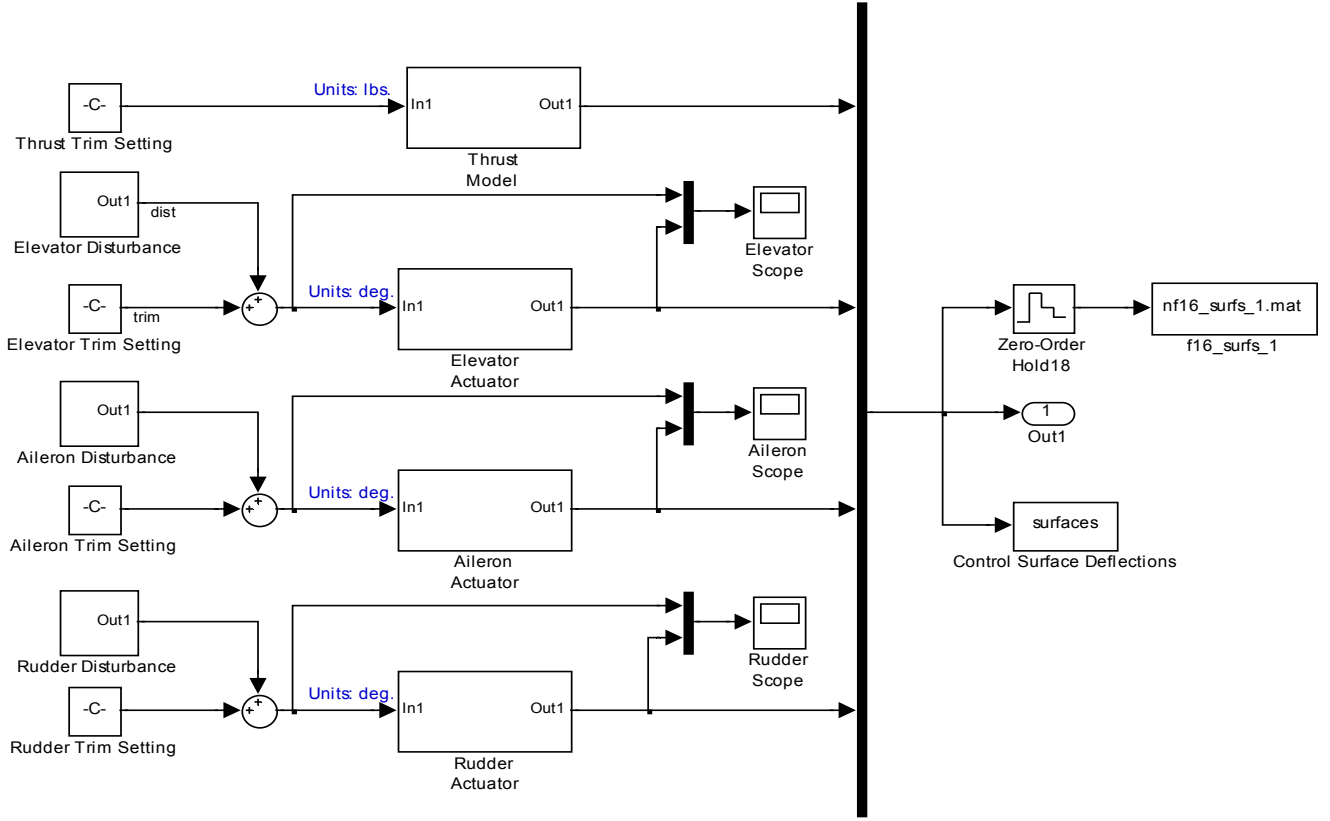


Figure 6. The Simulink model of the nonlinear F-16 aircraft cockpit of Figure 5.

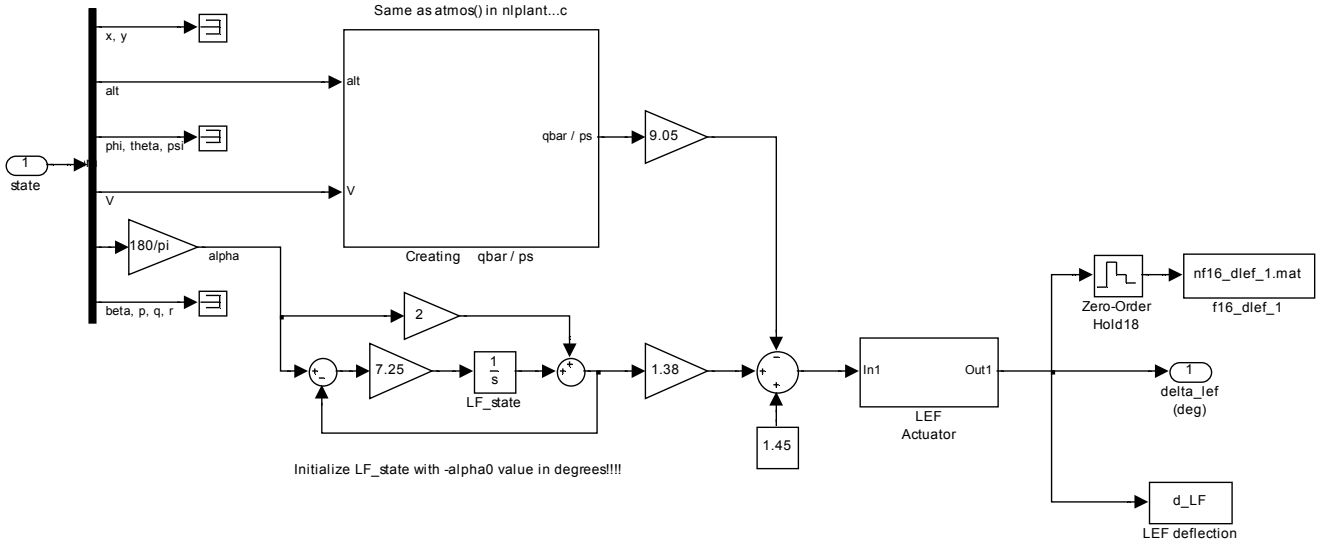


Figure 7. The Simulink model of the leading edge flap for the nonlinear F-16 aircraft.

3. Neural Network Model Identification and Neural Network-Based AGPC Control Scheme

3.1. The Nonlinear F-16 Aircraft Neural Network Model Identification Scheme

Assuming that at time k , the nonlinear F-16 aircraft can be

represented by as a p -input q -output nonlinear discrete-time system with disturbance term $\tilde{d}(k)$ by the following nonlinear autoregressive moving average with exogenous inputs (NARMAX) model [3–5]:

$$Y(k) = J \left[U(k-d), \dots, U(k-d-m), Y(k-1), \dots, Y(k-n) \right] + \tilde{d}(k) \quad (4)$$

where $J(\cdot, \cdot)$ is a nonlinear function of its arguments, and

$[U(k-d), \dots, U(k-d-m)]$ are the past input vector, $[Y(k-1), \dots, Y(k-n)]$ are the past output vector, $Y(k)$ is the current output, m and n are the number of past inputs and outputs respectively that define the order of the system, and d is time delay. The predictor form of (4) based on the information up to time $k-1$ can be expressed as [3–5]:

$$\hat{Y}(k | k-1, \theta(k)) = J[\varphi(k, \theta(k)), \theta^T(k)] \quad (5)$$

where $\varphi(k, \theta(k)) = [U(k-d), \dots, U(k-d-m), Y(k-1), \dots, Y(k-n), \varepsilon(k-1, \theta(k)), \dots, \varepsilon(k-n, \theta(k))]^T$ is the regression (state) vector, and $\theta(k)$ is an unknown parameter vector which must be selected such that $\hat{Y}(k | \theta(k)) \approx Y(k)$, $\varepsilon(k, \theta(k))$ is the error between (4) and (5) defined as

$$\varepsilon(k, \theta(k)) = Y(k) - \hat{Y}(k | k-1, \theta(k)) \quad (6)$$

Let a set of N input-output data pair obtained from prior system operation over NT period of time be defined:

$$Z^N = \{U(1), Y(1), \dots, U(N), Y(N)\}, \quad N = 1, 2, \dots \quad (7)$$

where T is the sampling time of the system outputs. Then, the minimization of (6) can be stated as follows:

$$\hat{\theta}(k) = \arg \min_{\theta(k)} J(Z^N, \varphi(k, \theta(k)), \theta(k)) \quad (8)$$

where $J(Z^N, \varphi(k, \theta(k)), \theta(k))$ is formulated as a mean square error (MSE) type cost function which can be stated as:

$$J(Z^N, \varphi(k, \theta(k)), \theta(k)) = \frac{1}{2N} \sum_{l=1}^N [\varepsilon(l, \theta(k))]^2 \quad (9)$$

The inclusion of $\theta(k)$ as an argument in $\varphi(k, \theta(k))$ is to account for the desired model $\hat{\theta}(k)$ dependency on $\tilde{d}(k)$. Thus, given an initial small random value of $\theta(k)$, m , n and (7), the model identification problem reduces to the minimization of (8) to obtain $\hat{\theta}(k)$.

The minimization of (8) is approached by considering $\hat{\theta}(k)$ as a neural network model. The complete NN model identification scheme based on the teacher-forcing method is illustrated in Figure 8 [3–5]. According to this scheme, the validated Simulink model of the nonlinear F-16 aircraft is placed in parallel with its NN model (in the dashed box) where the TDL (tapped delay line memory) are used to store temporal NN input information. The architecture of the “Neural Network Model” of Figure 8 is a dynamic

feedforward NN (DFNN) shown in Figure 9 [3–5]. The inputs to the DFNN model of Figure 9 are $\varphi_{n_a}(k) = [Y(k-1), \dots, Y(k-n)]^T$, $\varphi_{n_b}(k) = [U(k-d), \dots, U(k-d-m)]^T$ and $\varphi_{n_c}(k, \theta(k)) = [\varepsilon(k-1, \theta(k)), \dots, \varepsilon(k-n, \theta(k))]^T$ which are concatenated into $\varphi_l(k, \theta(k))$ as shown in Figure 9. The output of the NN model of Figure 8 in terms of the network parameters of Figure 9 is given as:

$$\hat{Y}(k | \hat{\theta}(k)) = F_i \left(\sum_{j=1}^{n_h} W_{i,j} f_j(\bar{a}) + W_{i,0} \right) \quad (10)$$

$$\bar{a} = \sum_{l=1}^{n_\varphi} w_{j,l} \varphi_l(k, \theta(k)) + w_{j,0}$$

where n_h and n_φ are the number of hidden neurons and number of regressors respectively; i is the number of outputs, $w_{j,l}$ and $W_{i,j}$ are the hidden and output weights respectively; $w_{j,0}$ and $W_{i,0}$ are the hidden and output biases; $F_i(\bar{b})$ is a linear activation function for the output layer and $f_j(\bar{a})$ is an hyperbolic tangent activation function for the hidden layer defined here as:

$$f_j(\bar{a}) = 1 - \frac{2}{e^{2\bar{a}} + 1} \quad (11)$$

The term “bias” is a weight acting on the input and clamped to 1. Here, $\hat{\theta}(k)$ is a collection of all network weights and biases in (10) in term of the matrices $w = \{w_{j,l}, w_{j,0}\}$ and $W = \{W_{i,j}, W_{i,0}\}$. Equation (10) is here referred to as NN NARMAX (NNNARMAX) model predictor for simplicity.

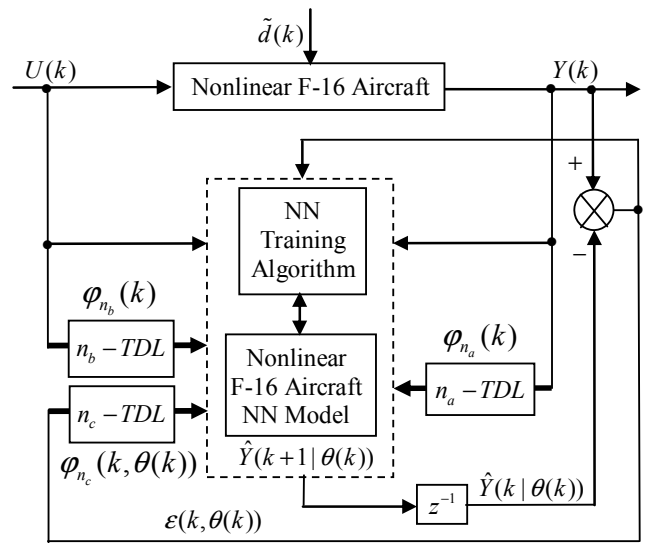


Figure 8. The nonlinear F-16 aircraft NN model identification scheme.

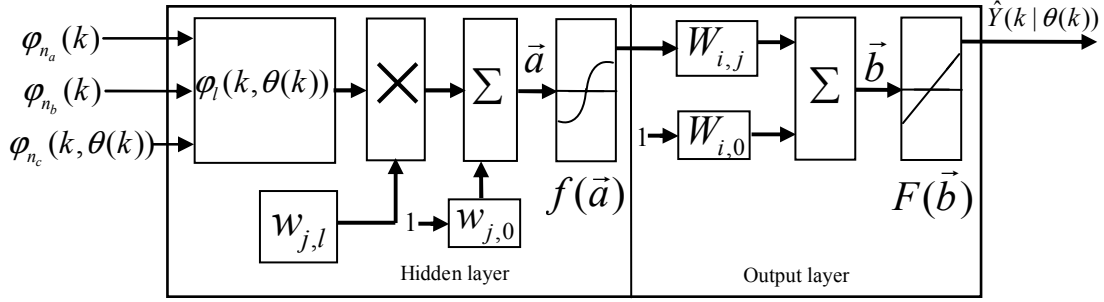


Figure 9. Architecture of the dynamic feedforward NN (DFNN) model.

The $\tilde{d}(k)$ in (4) is usually unknown but can be estimated as a covariance noise matrix using $\Gamma[\theta(k)] = E[\tilde{d}^T(k)\tilde{d}(k)]$ with an iterative algorithm described in [3, 5]. Thus, using $\Gamma[\theta(k)]$, Equation (9) becomes [3–5]:

$$J(\theta(k)) = \frac{1}{2N} \left(\sum_{l=1}^N \varepsilon[l, \theta(k)]^T \Gamma^{-1}[\theta(k)] \varepsilon[l, \theta(k)] + \theta^T(k) D \theta(k) \right) \quad (12)$$

where the second term in (12) is the regularization (weight decay) term [10] which has been introduced to reduce modeling errors, improve the robustness and performance of the two proposed training algorithms. $D = \alpha_d I = [\alpha_h \ \alpha_o] I$ is a penalty norm and also removes ill-conditioning, where I is an identity matrix, α_h and α_o are the weight decay values for the input-to-hidden and hidden-to-output layers respectively. Note that both $\hat{\Gamma}^{(j)}[\theta(k)]$ and D are adjusted simultaneously with $\theta(k)$ during network training and are used to update $\hat{\theta}(k)$ iteratively [3–5].

Several methods have been proposed in literatures for the minimization of (6) [3–5]. The formulation of (12) from (6) follows from the work of Akpan and co-workers where efficient training algorithms have been developed and validated [3–5]. The training algorithm adopted in this work is an online adaptive recursive least squares (ARLS) algorithm due to its proven efficiency [3–5].

3.2. The Neural Network-Based AGPC Control Scheme for the Nonlinear F-16 Aircraft

The structure of the NN-based model identification and AGPC control scheme is shown in Figure 10; where $R'(k)$, $E(k)$, $U(k)$, $Y(k)$, $\hat{Y}(k)$, $\hat{Y}(k+\eta|k)$, $\tilde{d}(k)$ and $z^{-\eta}$ are the desired reference signal, prediction error, control input, system output, η step-delay prediction model output, η step-ahead predicted output, noise/input disturbances and η step-delay operator respectively and k is the number of samples based on the new measurement data sample.

As in basic MPC scheme [4, 5, 9–11, 32], the prediction errors $E(k)$ between the process model and the prediction model are compensated by filtering the reference signal using a first-order low-pass digital filter defined here as:

$$R(k) = \frac{B_m}{A_m} R'(k) \quad (13)$$

where $R'(k)$ and $R(k)$ are the desired reference and the filtered reference signals respectively; A_m and B_m are the denominator and numerator polynomials of the filter. In this way, the AGPC is designed, in part, based on the filter tracking error capability; where A_m and B_m serves as tuning parameters used to improve the robustness and internal stability of the AGPC algorithm respectively.

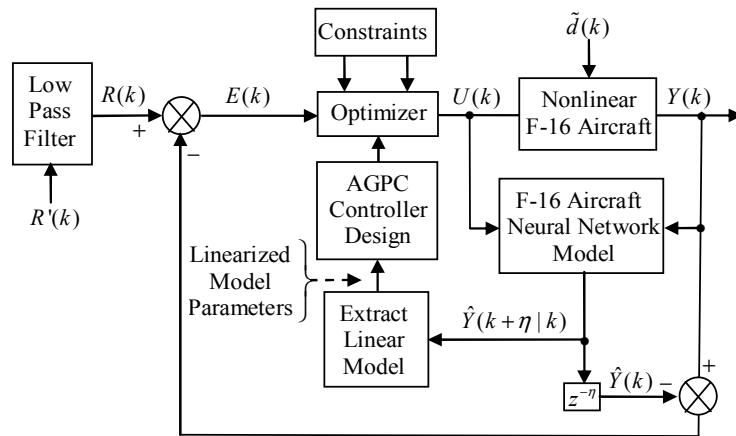


Figure 10. The NN-based AGPC control scheme with the nonlinear F-16 aircraft and its neural network model.

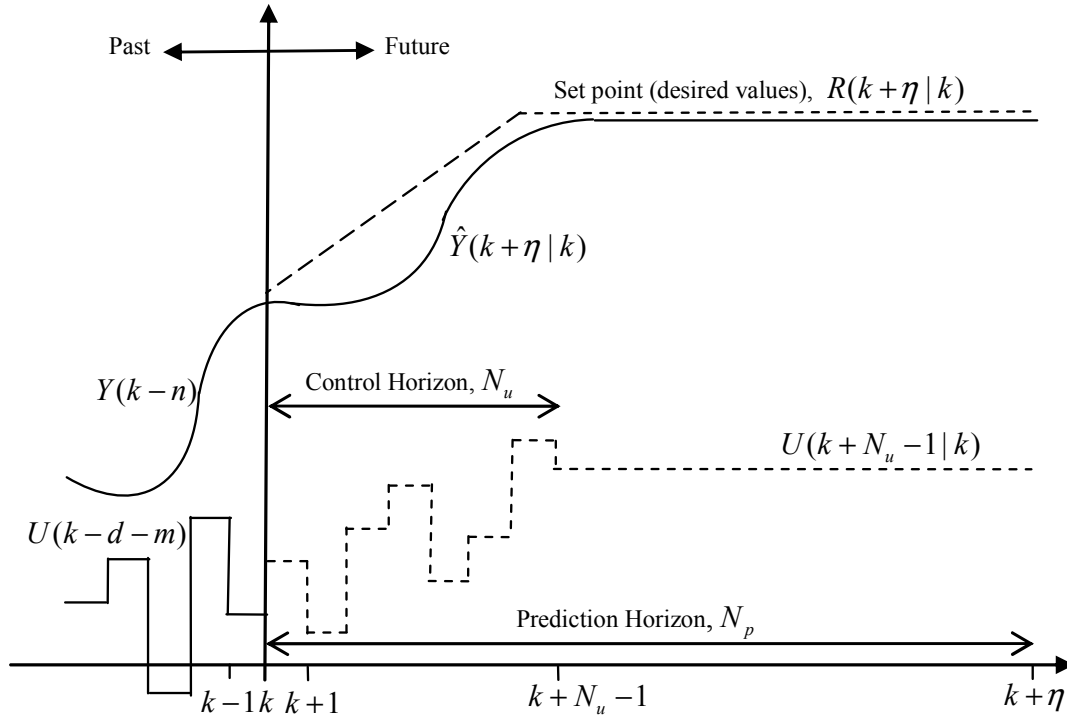


Figure 11. The AGPC control strategy.

The AGPC control strategy is based on a receding horizon principle illustrated in Figure 11 and depends on an explicit model of the system. The AGPC scheme is summarized as follows:

At the current sampling time k , the NN model predictor uses the past m -inputs, n -outputs and the current system information to identify the nonlinear discrete-time NN model of the nonlinear F-16 aircraft.

Assuming that the identified NN model is stable, proper and deterministic; then the AGPC algorithm uses the linearized model parameters of the identified nonlinear NN model to accurately predict the current system output $\hat{Y}(k)$ at that same sample time instant k .

At time $k + N_u - 1$, the AGPC algorithm calculates a sequence of control inputs $U(k + N_u - 1 | k)$ consisting of the current $U(k | k)$ and future inputs $U(N_u - 1 | k)$. The current input $U(k) = U(k | k)$ is held constant after N_u control moves; where N_u is the maximum control horizon. The input $U(k)$ is calculated such that a set of $\hat{Y}(k + \eta | k)$ approaches the desired reference signal in an optimal manner over a specified prediction horizon $\eta \in [N_d, N_p]$; where N_d and N_p are the minimum and maximum prediction horizons respectively.

The predicted values are used to calculate the control signals by minimizing an objective function of the form:

$$\hat{J}(U(k)) = \left[R(k) - \hat{Y}(k) \right]^T \kappa \left[R(k) - \hat{Y}(k) \right] + \tilde{U}^T(k) \rho \tilde{U}(k) \quad (14)$$

subject to the constraints

$$\Delta U(k + \eta) = 0, N_u \leq \eta \leq N_p - N_d \quad (15)$$

$$\Delta U_{\min} \leq \Delta U(k) \leq \Delta U_{\max}, Y_{\min} \leq Y(k) \leq Y_{\max} \quad (16)$$

where

$$R(k) \triangleq [R(k + N_d) \dots R(k + N_p)]^T$$

$$\hat{Y}(k) \triangleq [\hat{Y}(k + N_d | k) \dots \hat{Y}(k + N_p | k)]^T$$

$$E(k) = [R(k) - \hat{Y}(k)] \triangleq [E(k + N_d | k) \dots E(k + N_p | k)]^T$$

$$\tilde{U}(k) \triangleq [\Delta U(k) \dots \Delta U(k + N_u - N_d)]^T$$

where ΔU is the change in control signal; κ and ρ are two weighting matrices penalizing changes on $\hat{Y}(k)$ and $U(k)$ in (14).

Although a sequence of N_u moves is calculated at each sampling instant, only the first control move $U(k) = U(k | k)$ is actually implemented and applied to control the process. The remaining control signals are not applied because at the next sampling instant $k = k + 1$ a new output $Y(k + 1)$ is known based on new measurements. The AGPC strategy enters a new optimization loop while the remaining control signals $U(N_u - 1 | k)$ are used to initialize the optimizer. This is indeed the receding horizon principle inherent in MPC strategy.

The AGPC algorithm used in this work is taken from the [4, 5], where it has been demonstrated to be suitable for the

auto-pilot control of the nonlinear F-16 aircraft but with an average computation time of approximately 6.1048 seconds at each sampling time which violates the 0.5 seconds sampling time of the nonlinear F-16 aircraft [4]. Hence, the effort in this work is directed towards reducing the computation time at each sampling instant by modeling, synthesizing and mapping the AGPC algorithm to Virtex-5 FX70T ML507 FPGA embedded system development board.

4. Implementation of the Model Identification and AGPC Algorithms for the Nonlinear F-16 Aircraft

4.1. Neural Network Identification of the Nonlinear F-16 Aircraft Model

The neural network model predictor which is based on a nonlinear autoregressive with moving average exogenous input (NNARMAX) discussed in Section 3.1 is considered for modeling the nonlinear F-16 aircraft. The neural network identification scheme of Figure 8 with the architecture of Figure 9 are used here. The input vector to the neural network consists of the regression (state) vectors $\varphi_{n_a}(k)$, $\varphi_{n_b}(k)$ and $\varphi_{n_c}(k, \theta(k))$ which are concatenated into $\varphi_{NNARMAX}(k, \theta(k))$ as shown in Figure 9. All these vectors are defined by the following relationships:

$$\varphi_{n_a}(k) = \begin{bmatrix} P_N(k-n_a) & P_E(k-n_a) & h(k-n_a) \\ \phi(k-n_a) & \vartheta(k-n_a) & \psi(k-n_a) \\ V_T(k-n_a) & \alpha(k-n_a) & \beta(k-n_a) \\ p(k-n_a) & q(k-n_a) & r(k-n_a) \\ A_{nx}(k-n_a) & A_{ny}(k-n_a) & A_{nz}(k-n_a) \\ M(k-n_a) & \bar{q}(k-n_a) & p_s(k-n_a) \end{bmatrix}^T \quad (17)$$

$$\varphi_{n_b}(k) = \begin{bmatrix} u_{\delta_a}(k-n_b) & u_{\delta_e}(k-n_b) \\ u_{\delta_r}(k-n_b) & u_{\delta_{LEF}}(k-n_b) \end{bmatrix}^T \quad (18)$$

$$\varphi_{n_c}(k, \theta(k)) = \begin{bmatrix} \varepsilon_{P_N}(k-n_c, \theta(k)) & \varepsilon_{P_E}(k-n_c, \theta(k)) \\ \varepsilon_h(k-n_c, \theta(k)) & \varepsilon_\phi(k-n_c, \theta(k)) \\ \varepsilon_{\vartheta}(k-n_c, \theta(k)) & \varepsilon_\psi(k-n_c, \theta(k)) \\ \varepsilon_{V_T}(k-n_c, \theta(k)) & \varepsilon_\alpha(k-n_c, \theta(k)) \\ \varepsilon_\beta(k-n_c, \theta(k)) & \varepsilon_p(k-n_c, \theta(k)) \\ \varepsilon_q(k-n_c, \theta(k)) & \varepsilon_r(k-n_c, \theta(k)) \\ \varepsilon_{A_{nx}}(k-n_c, \theta(k)) & \varepsilon_{A_{ny}}(k-n_c, \theta(k)) \\ \varepsilon_{A_{nz}}(k-n_c, \theta(k)) & \varepsilon_M(k-n_c, \theta(k)) \\ \varepsilon_{\bar{q}}(k-n_c, \theta(k)) & \varepsilon_{p_s}(k-n_c, \theta(k)) \end{bmatrix}^T \quad (19)$$

$$\varphi_{NNARMAX}(k, \theta(k)) = [\varphi_{n_a}(k); \varphi_{n_b}(k); \varphi_{n_c}(k, \theta(k))] \quad (20)$$

where $u_{\delta_a}(k)$, $u_{\delta_e}(k)$, $u_{\delta_r}(k)$, $u_{\delta_{LEF}}(k)$ and $u_{\delta_{LEF}}(k)$ are the aileron deflection, elevator deflection, rudder deflection, throttle setting and the deflection of the leading edge flap respectively; $V_T(k)$, $\alpha(k)$ and $\beta(k)$ are the velocity, angle of attack and angle of sideslip respectively; $\phi(k)$, $\vartheta(k)$ and $\psi(k)$ are the Euler angles; $p(k)$, $q(k)$ and $r(k)$ are the angular rates.

Although, the actual outputs of the neural network are the predicted values of the twelve states given by:

$$\hat{Y}(k) = \begin{bmatrix} \hat{P}_N(k) & \hat{P}_E(k) & \hat{h}(k) & \hat{\phi}(k) & \hat{\vartheta}(k) \\ \hat{\psi}(k) & \hat{V}_T(k) & \hat{\alpha}(k) & \hat{\beta}(k) & \hat{p}(k) \\ \hat{q}(k) & \hat{r}(k) & \hat{A}_{nx}(k) & \hat{A}_{ny}(k) & \\ \hat{A}_{nz}(k) & \hat{M}(k) & \hat{\bar{q}}(k) & \hat{p}_s(k) \end{bmatrix}^T \quad (21)$$

which are also used for the error predictions of (19), for simplicity reasons, the results of the simulations that follow are only for the predicted values of the three angular rates and the throttle setting given by:

$$\hat{Y}(k) = [\hat{p}(k) \quad \hat{q}(k) \quad \hat{r}(k) \quad \hat{i}(k)]^T \quad (22)$$

where $\hat{p}(k)$, $\hat{q}(k)$, $\hat{r}(k)$, and $\hat{i}(k)$ are the predicted values of the roll rate, pitch rate, yaw rate and the throttle setting respectively. Although the fifth output of the controller; that is, the deflection of the leading edge flap δ_{LEF} , contributes to the performance of the F-16 aircraft, it is left out in the simulation results since it is not directly available for manipulation by the pilot but depends on α , q and p_s as discussed and illustrated in Figure 2 [1, 2, 5, 6].

The neural network model identification problem here is to train a neural network in order to determine the optimal parameters of the network which will provide the same values of the controlled variables with those obtained from the aircraft when both neural network and aircraft are subjected to the same input stimuli. Then, the trained network will be employed as the model of the aircraft on which the computations of the control actions will be based at each sampling instant in this will be also the model that will be upgraded each time a new set of input-output data become available from the actual operation of the aircraft. The disturbances considered here are variations in the parameters of the validated nonlinear F-16 aircraft model build by first principles.

4.2. Neural Network Training for Nonlinear F-16 Aircraft NNARMAX Model Identification

The input vector to the neural network is the regression vector $\varphi_{NNARMAX}(k, \theta(k))$ defined by (20). The regressors of the moving average input vector $\varphi_{n_c}(k, \theta(k))$, are not usually

known in advance and it is initialized to small positive random matrix of dimension n_c by n_c . The outputs of the neural network are predicted values of $\hat{Y}(k)$ given by (19). However, as discussed earlier, the simulation results for the output predictions presented here are for the roll rate $\hat{p}(k)$, pitch rate $\hat{q}(k)$ and yaw rate $\hat{r}(k)$ as defined in (22).

For assessing the convergence performance of the network, the network is trained for $\tau = 100$ Epoch with the following selected parameters: $p = 4$, $q = 18$, $n_a = 2$, $n_b = 2$, $n_c = 2$, $n_\phi = 80$, $n_h = 10$, $n_o = 18$, $\alpha_h = 1e-7$ and $\alpha_o = 1e-6$. The four design parameters for adaptive recursive least squares (ARLS) algorithm are selected to be: $\alpha = 0.5$, $\beta = 5e-3$, $\delta' = 1e-5$ and $\pi = 0.99$ resulting to $\gamma = 0.0101$. The initial values for \bar{e}_{min} and \bar{e}_{max} are equal to 0.0102 and 1.0106e+3 respectively and were evaluated [3–5]. Thus, the ratio $\bar{e}_{min}/\bar{e}_{max}$ which equals 9.9018e+4 is in agreement that the parameters are well selected [5].

However, the training data are scaled to unit variance using their mean values and standard deviations according to the following equations [5]:

$$\left. \begin{aligned} U^{(s)}(k) &= \frac{U(k) - \bar{U}(k)}{\sigma_{U(k)}} \\ Y^{(s)}(k) &= \frac{Y(k) - \bar{Y}(k)}{\sigma_{Y(k)}} \end{aligned} \right\} \quad (23)$$

where $\bar{U}(k)$, $\bar{Y}(k)$ and $\sigma_{U(k)}$, $\sigma_{Y(k)}$ are the mean and standard deviation of the input and output training data pair; and $U^{(s)}(k)$ and $Y^{(s)}(k)$ are the scaled inputs and outputs respectively. Also, after the network training, the joint weights are rescaled according to the expression

$$\hat{Y}(k, \hat{\theta}(k)) = \hat{Y}(k, \hat{\theta}(k))\sigma_{Y(k)} + \bar{Y}(k) \quad (24)$$

so that the trained network can work with other unscaled validation data and test data not used for training. However, for notational convenience, $U(k) = U^{(s)}(k)$ and $Y(k) = Y^{(s)}(k)$ shall be used.

The 4,000 training data is first scaled according to Equation (23) and the network is trained for $\tau = 100$ epochs using the ARLS algorithm discussed in Section 3.1. After network training, the trained network is again rescaled using Equation (24), so that the network trained for the nonlinear F-16 aircraft can work with unscaled nonlinear F-16 aircraft data.

The evaluation of the identified nonlinear F-16 aircraft NNARMAX model trained with ARLS algorithm examined here as summarized in Table 2.

The computation times for the network trained with 100 epochs are shown in the first row of Table 2 is 2.9859e+2 seconds. The mean square error (MSE) and the minimum performance indexes for the network trained with the ARLS are given in the second and third rows of Table 2 as 1.1194e-1 and 2.0910e-6 respectively. The relatively small values of the

mean square error (MSE) and the minimum performance indexes indicate that the ARLS captures the nonlinear dynamics of the nonlinear F-16 aircraft.

Table 2. Summary of training results using the ARLS algorithm for nonlinear F-16 aircraft.

S/N	Evaluation Criteria	Values
1	Computation time for model identification (sec)	2.9859e+2
2	Mean value of the Mean square error (MSE)	1.1194e-1
3	Minimum performance index	2.0910e-6
4	Mean error of one-step ahead prediction of training data	8.4186e-5
5	Mean error of one-step prediction of validation data	4.7908e-2
6	Mean value of 5-step ahead prediction error	1.6524e-2
7	Akaike's final prediction error (AFPE) estimate	2.5060e-2

4.3. Validation of the Trained Network for Modeling the Nonlinear F-16 Aircraft

Generally, the process of checking if a trained network predicts correctly both data that were used for training and unknown data that were not used during training is called network validation. In the following, the validation by different methods of the network that models the nonlinear F-16 aircraft trained by the ARLS algorithm is explained. This validation has been made by the use of scaled and unscaled data as well as with 400 validation data obtained from the experiments and the open-loop simulations discussed in Section 2.2.

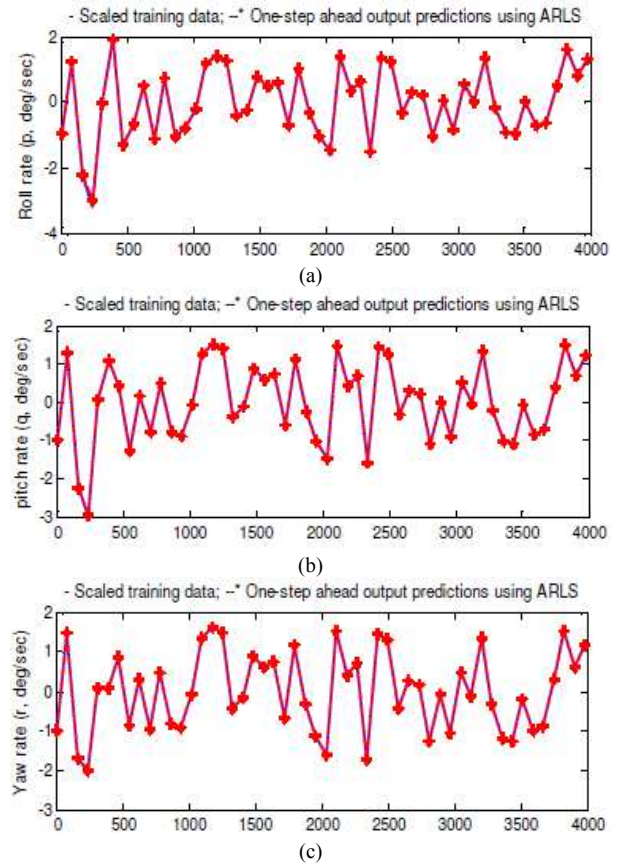


Figure 12. Output predictions of the scaled training data using the network trained with ARLS algorithm: (a) roll rate prediction, (b) pitch rate prediction and (c) yaw rate prediction.

4.3.1. Trained Network Validation by the One-Step Ahead Predictions Simulation

The one-step ahead prediction method makes an assessment of the errors between the training data obtained from the open-loop simulation of the differential equations model and the one-step ahead predictions of the trained network.

The comparison of the one-step ahead predictions of the scaled training data (blue -) against the trained network output predictions (red --*) using the ARLS algorithm are shown in Figure 12(a)–(c) for the roll rate (\hat{p}), pitch rate (\hat{q}) and yaw rate (\hat{r}) respectively.

The one-step ahead prediction errors for the scaled training data for 100 epochs when the network is trained by using the ARLS is given in the fourth line of Table 2 as $8.4186\text{e-}5$. It can be seen in Figure 12(a)–(c) that the network predictions of the roll rate (\hat{p}), pitch rate (\hat{q}) and yaw rate (\hat{r}) training data closely match the original data used for the network training. The small one-step ahead prediction error is an indication that the trained network approximates the nonlinear dynamics of the nonlinear F-16 aircraft to an appreciable degree of accuracy. This is further justified by the small mean values of the MSE and the performance indexes obtained by using the ARLS.

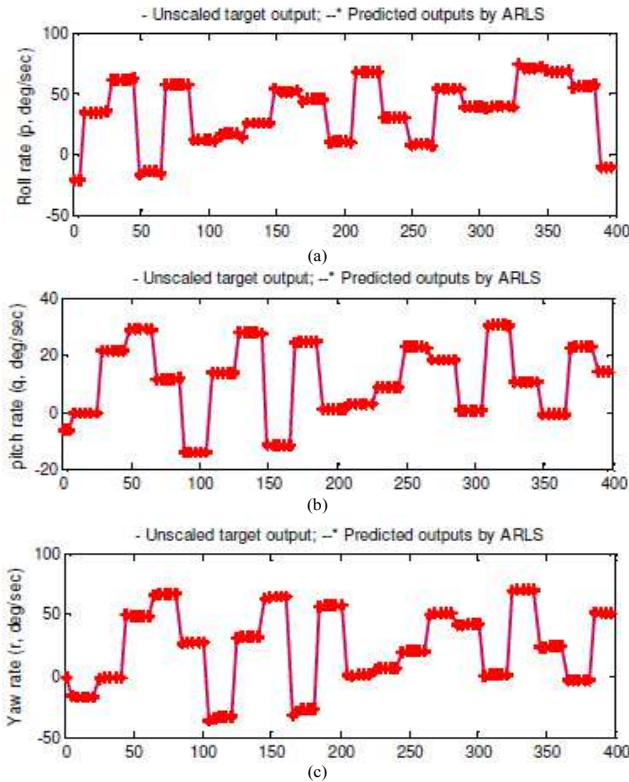


Figure 13. Output predictions of the unscaled validation data using the network trained with ARLS algorithm: (a) roll rate prediction, (b) pitch rate prediction and (c) yaw rate prediction.

Furthermore, the trained network by the ARLS algorithm was validated with 400 unscaled data obtained directly from experiments from the operation of the nonlinear F-16 aircraft. The comparison of the trained network predictions (red --*)

with these validation data (blue -) are shown in Figure 13(a)–(c) for the roll rate (\hat{p}), pitch rate (\hat{q}) and yaw rate (\hat{r}) respectively. The prediction accuracies of the unscaled validation data were assessed by the computed mean prediction errors shown in the fifth line of Table 2 as $4.7905\text{e-}2$. This small prediction error for the unscaled validation data given by Figure 13 as well as the prediction errors in Table 2 justify that the network mimics satisfactorily the dynamics of the nonlinear F-16 aircraft.

4.3.2. K-Step Ahead Prediction Simulations

The results of the K -step ahead output predictions (red --*) using the K -step ahead prediction validation method for 5-step ahead output predictions ($K = 5$) compared with the unscaled training data (blue -) are shown in Figure 14(a)–(c) for the roll rate (\hat{p}), pitch rate (\hat{q}) and yaw rate (\hat{r}) K -step ahead output predictions based on the networks trained with the ARLS algorithm. The value of $K = 5$ is chosen since it is a typical value used in most MPC applications [9–11, 32].

The mean value of the 5-step ahead prediction error (MVPE) is $1.6506\text{e-}2$ based on the ARLS algorithm as shown in the sixth row in Table 2. The small mean values of the 5-step ahead prediction error (MVPE) are indications that the trained network approximates the dynamics of the nonlinear F-16 aircraft to an appreciable degree of accuracy.

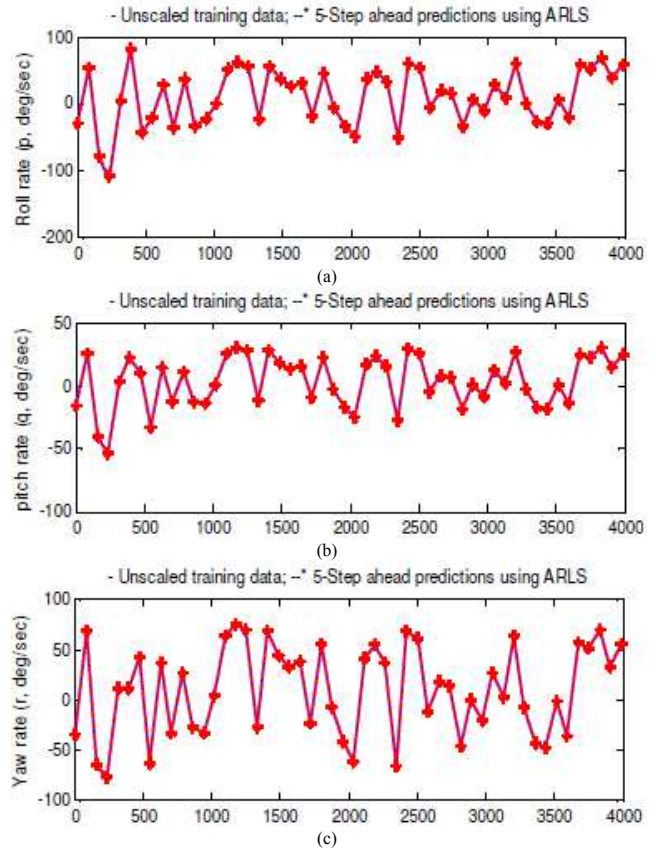


Figure 14. Comparison of 5-step ahead output predictions (red--*) with the original unscaled training data (blue-) using the network trained with ARLS algorithm: (a) roll rate prediction, (b) pitch rate prediction and (c) yaw rate prediction.

However, the small output predictions errors obtained by the networks using ARLS algorithm is acceptable but the further verification of the ARLS algorithm for online identification and adaptive control will be investigated to further justify the model accuracy.

4.3.3. The Akaike's Final Prediction Error (AFPE) Estimates

The implementation of the AFPE algorithm resulted to the estimates which is given in the last row of Table 2 as $2.5060e-2$. This implementation concerns the network that is trained with ARLS algorithm with multiple weight decay for the regularized criterion. These small values of the AFPE estimates indicate that the trained network captures the underlying dynamics of the nonlinear F-16 aircraft and is not over-trained [42]. This in turn implies that optimal network parameters have been selected including the weight decay parameters.

4.4. Online Closed-Loop Model-Based Adaptive Control of the Nonlinear F-16 Aircraft

Besides the training of the neural network model with static data taken during the open loop simulation experiments, it is of interest to observe the online behaviour of the ARLS network training algorithm in closed-loop with the AGPC control of the nonlinear F-16 aircraft flight control variables.

Table 3. Constraints for the nonlinear F-16 aircraft.

S/N	Nonlinear F-16 Aircraft Constraint	Aileron	Elevator	Rudder
1	Minimum control input (U_{\min})	-21.5	-25	-30
2	Maximum control input (U_{\max})	21.5	25	30
3	Minimum predicted output (Y_{\min})	-80	-60	-120
4	Maximum predicted output (Y_{\max})	80	60	120
5	Desired reference signal (R')	± 80	± 60	± 120

Table 4. Tuning parameters for the AGPC controller.

S/N	Tuning Parameters	Aileron	Elevator	Rudder
1	Initial control input (ICI, U)	11	11	11
2	Initial predicted output (IPO, Y)	0	0	0
3	N_d	1	1	1
4	N_a	2	2	2
5	N_p	4	4	4
6	κ	2	1	1.3
7	ρ	1	0.8	0.8
8	A_m	[1 -0.7]	[1.05 -0.7]	[1 -0.7]
9	B_m	[0 0.3]	[0 0.3]	[0 0.3]

Initially, the NNARMAX model of the nonlinear F-16 aircraft is identified and validated as in the Section 4.4. The AGPC controller is then simulated subject to the constraints given in Table 3 and tuned using the NNARMAX model of the nonlinear F-16 aircraft. The obtained optimal tuning parameters are given in Table 4. Next, the nonlinear F-16 aircraft model is placed in closed-loop with the NNARMAX model identification scheme of Figure 8 based on the ARLS algorithm and the AGPC controller of Figure 10. At each sampling instant, a new input-output data is obtained from the first principle Simulink model of the nonlinear F-16 aircraft, a neural network model is identified with 100 Epochs and the AGPC controller is

designed and implemented online for 160 samples.

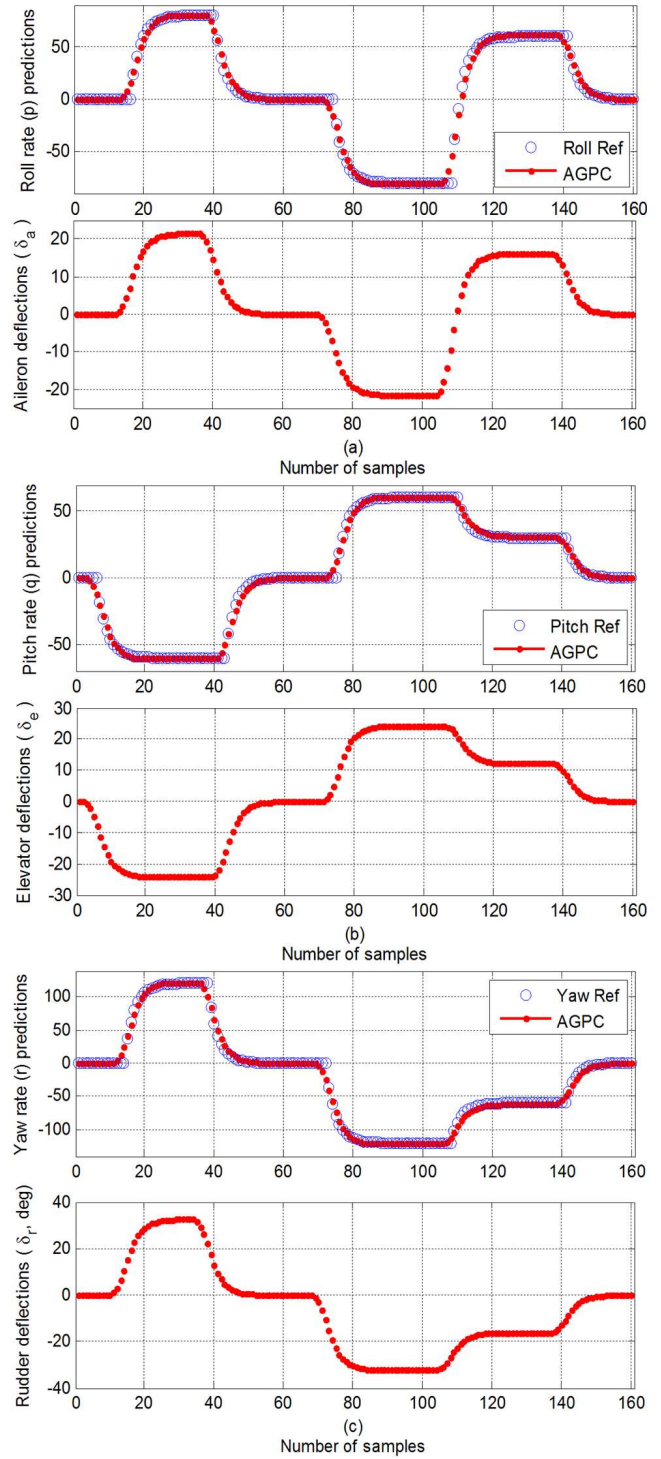


Figure 15. Closed-loop responses of the controlled variables (top) and the manipulated variables (bottom) when NN is trained with ARLS algorithm for 100 epochs at each sampling time: (a) roll rate and aileron deflection, (b) pitch rate and elevator deflection, and (c) yaw rate deflection.

The closed-loop simulation results for the desired outputs (controlled variables), namely: the roll rate (\hat{p}), pitch rate (\hat{q}), yaw rate (\hat{r}) are shown in the top part of Figure 15(a)–(c) while the flight control variables (manipulated inputs),

namely: the aileron deflection δ_a , elevator deflection δ_e and rudder deflection δ_r are shown in the lower part of Figure 15(a)–(c).

The excellent tracking of the roll, pitch and yaw rates by the closed-loop scheme are evident in Figure 15(a)–(c) which justifies the effectiveness and efficiency of the ARLS algorithm and AGPC controller. The closed-loop scheme shows better control performance with no overshoot or non-minimum behaviour in tracking the desired reference trajectories with relatively small energy consumption required for the manipulated variables (control inputs).

4.5. Computation Time for the Neural Network Model Identification and Adaptive Control of the Nonlinear F-16 Aircraft Auto-Pilot System

The NNARMAX model identification scheme and AGPC control strategies above were implemented on an Intel® Core™2 CPU running at 1.86GHz using the MATLAB “*parfor*” command available in the MATLAB Distributed and Parallel Toolbox. The “*parfor*” command implements the loop specified by the command in parallel and it uses the four Intel® processors available on the computer system. This MATLAB facility allows the utilization of the four processors available on the computer for the implementation of the identification and control algorithm at each sampling time step. The “*parfor*” is initialized using the MATLAB “*matlabpool open x*” command, where *x* specifies the number of processor(s) to be used in the computation. Of course, after the computation, the parallel session is closed using the command “*matlabpool close*”.

The plot of the computation time at each time sample is shown in Figure 16. The average computation time for the NNARMAX model identification and AGPC control closed-loop from Figure 16 is calculated to be 6.1048 seconds which violates the 0.5 seconds sampling time of the nonlinear F-16 aircraft.

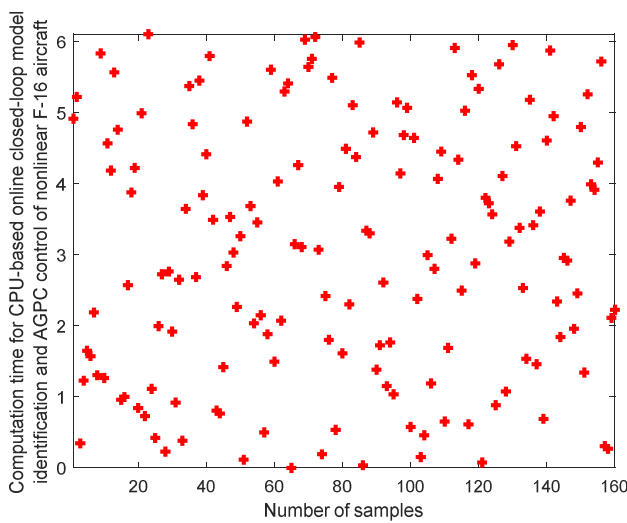


Figure 16. Computation time for the parallel implementation of the NNARMAX model identification and AGPC control closed-loop simulation of the nonlinear F-16 auto-pilot control system at each time sample per 1 clock pulse.

5. Modeling, Synthesis, Verification and Hardware-in-the-Loop Hardware Co-Simulation of NN-Based AGPC

5.1. Overview of the Xilinx Model-Based Design Flow of an Embedded System

The integration of Simulink and MATLAB from The MathWorks, Inc. [37] and the Xilinx FPGA design suite of tools [38] now allows the development of model-based design of a system on a FPGA. Recently a lot of research work has been carried for the implementation of model predictive control (MPC) algorithms on a FPGA [5, 12–20]. They involve the solutions of a computationally intensive online optimization problem at a very short sampling time interval. Additionally, as nonlinearity is the characteristic of many industrial systems, FPGA implementation of neural network algorithms, which seems to be an efficient method for modeling the dynamics of nonlinear systems is investigated in this paper.

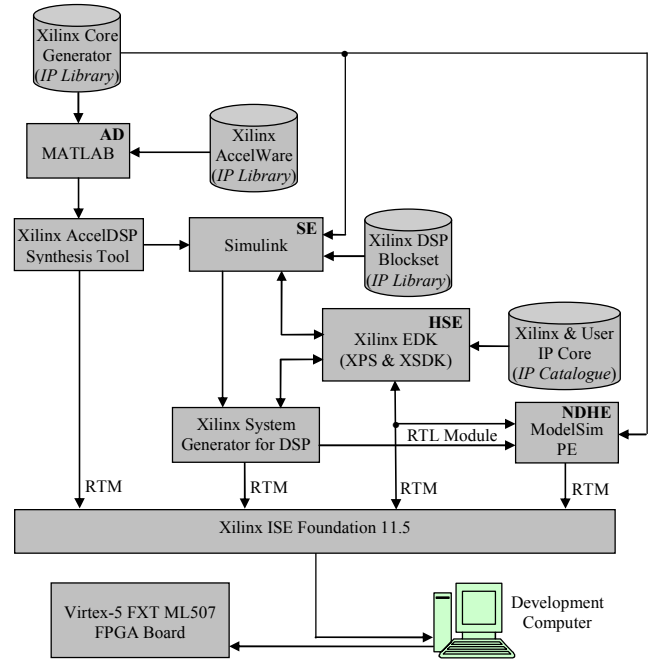


Figure 17. Embedded system design flow: IP – Intellectual Property, AD – algorithm developer, SE – system engineer, HSE – hardware/software engineer, NDSPHE – Non-DSP hardware engineer, EDK – Embedded Development Kit, XPS – Xilinx Platform Studio, XSDK – Xilinx Software Development Kit, RTM – RTL Top-Level Module, ISE – Integrated Software Environment.

A flow diagram showing a model-based design methodology is shown in Figure 17. A related but reduced architecture from an ASIC point of view has been reported by Meloni and co-workers [39]. As shown in Figure 17, four design approaches can be identified for implementing an FPGA-based design. One is from the point of view of an algorithm developer (AD), another from the point of view of a system engineer (SE) or a hardware/software engineer (HSE) or a non-DSP hardware engineer (NDHE). In this work, the

first three FPGA-based design and implementation approaches are presented from an AD, a SE and a HSE view points using model-based design methodologies. The term “model-based design” here refers to design problems formulated as algorithms and developed using MATLAB and Simulink from The MathWorks [37].

5.2. Model-Based Design Approach for the FPGA Implementation of the NN-Based AGPC Algorithm

The model-based approach is proposed here as the technique to be used for the efficient realization and implementation of the AGPC algorithm on the FPGA board. The term “model-based” is based on the fact that:

A hardware model of the AGPC algorithm is first realized by synthesizing the AGPC algorithm expressed and implemented as MATLAB programs using the Xilinx AccelDSP synthesis tool.

Simulink is then used to model the resulting AGPC algorithmic hardware model as a complete system using additional intellectual property (IP) cores from the Xilinx System Generator for DSP block library, and A hardware model called the Hardware Co-Simulation (HW-CoSim) block model that can encrypt the complete System Generator model of the AGPC algorithm is then generated to perform a *priori* FPGA-in-the-loop co-simulations using the actual nonlinear F-16 aircraft model available in the MATLAB/Simulink environment. By this way, how the complete AGPC algorithmic hardware model will perform when programmed into the FPGA can be evaluated right from the Simulink environment via the Xilinx System Generator for DSP, and all design modifications can be made at this point if the simulation results of the synthesized hardware model performance does not meet the desired design requirements.

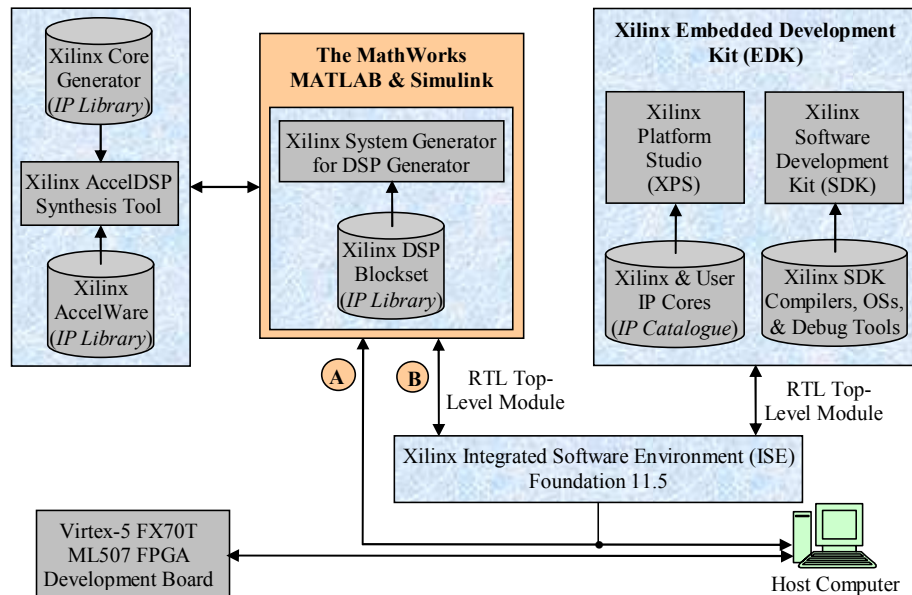


Figure 18. The block diagram for the proposed model-based design flow for the FPGA implementation of the AGPC algorithm on Virtex-5 FX70T ML507 FPGA development board.

The simplified block diagram for the proposed model-based approach for the FPGA implementation of the AGPC algorithm is shown in Figure 18. The block diagram of Figure 18 consists of five distinct blocks as used in this work are:

The MATLAB/Simulink from The MathWorks is used for modeling and verifying the algorithmic and model-based designs.

The Xilinx AccelDSP which is used for modeling and synthesizing MATLAB-based algorithms to generate a synthesizable hardware model of the algorithm,

The Xilinx System Generator for DSP, which is a sub-set of MATLAB/Simulink, first uses the generated hardware from the AccelDSP design flow to create a complete synthesizable model of the AGPC algorithm, create a hardware co-simulation (HW Co-Sim) block for FPGA-in-the-loop simulation for performance verification of the System Generator model, and finally exports the System Generator

model as a *pcore* for integration with a pre-designed PowerPCTTM440 embedded processor system,

The Xilinx embedded development kit (EDK) is used to design an embedded processor system. The embedded processor proposed for use in this work is the PowerPCTTM440 hard processor core. Note that there is no link between MATLAB/Simulink or System Generator for DSP with the Xilinx EDK; rather the AGPC *pcore* generated by the System Generator for DSP is copied manually to the embedded processor directory and integrated with a pre-designed embedded PowerPCTTM440 processor system in the XPS environment of the within the EDK.

The Xilinx integrated software environment (ISE) Foundation uses the register-transfer-level (RTL) top-level module of the design to generate an optimized bitstream for programming the Virtex-5 FX70T ML507 FPGA development board via the serial ports of the host computer and the FPGA board.

Note that the circled symbol (A) is used to illustrate the System Generator path for FPGA-in-the-loop using the hardware co-simulation block without using the Xilinx ISE as a gateway. However, it calls and uses the Xilinx ISE Foundation during the compilation, optimization routines and generation of the hardware co-simulation block as indicated by the circled symbol (B). Although the host computer is not counted as part of the five blocks, it is the main development platform upon which all implementations and simulations are performed.

In the remaining parts of the paper, the synthesis, modeling, implementation, verification and performance evaluation of the FPGA implementation of the AGPC algorithm on the Xilinx Virtex®-5 FX70T FPGA board are given.

5.3. The Block Diagram for the Closed-Loop Implementation of the Nonlinear F-16 Aircraft Auto-Pilot Control on Virtex-5 FX70T ML507 FPGA

In this work, the FPGA realization and implementation of the AGPC algorithm is investigated using Virtex-5 FX70T FPGA development board [5]. The verification and performance evaluation of the proposed FPGA implementation of the AGPC

algorithm on the Virtex-5 FX70T ML507 FPGA development board is performed in closed-loop with the nonlinear F-16 aircraft. In this study, the NNARMAX model identification scheme based on ARLS algorithm is implemented on the host development computer, which will also simulate the validated Simulink model of the nonlinear F-16 aircraft in closed-loop Virtex-5 FPGA at each sampling time. The proposed strategy for the FPGA-in-the-loop implementation, hardware co-simulation verification and performance evaluation of the AGPC algorithm is shown in Figure 19. In this figure, the host development computer is shown on the right; the Virtex-5 FX70T ML507 FPGA development board is shown in the middle; and the display monitor is shown on the left. The neural network model identification scheme based on the ARLS algorithm and the nonlinear F-16 are configured and programmed to run in MATLAB and Simulink. As in Section 4.4, $U(k)$ is the control input vector to the F-16 aircraft, $Y(k)$ is the output response of the F-16 aircraft, $\tilde{d}(k)$ is the disturbances affecting the F-16 aircraft, and $\hat{\theta}(k)$ is the identified neural network (NNARMAX) model of the F-16 aircraft.

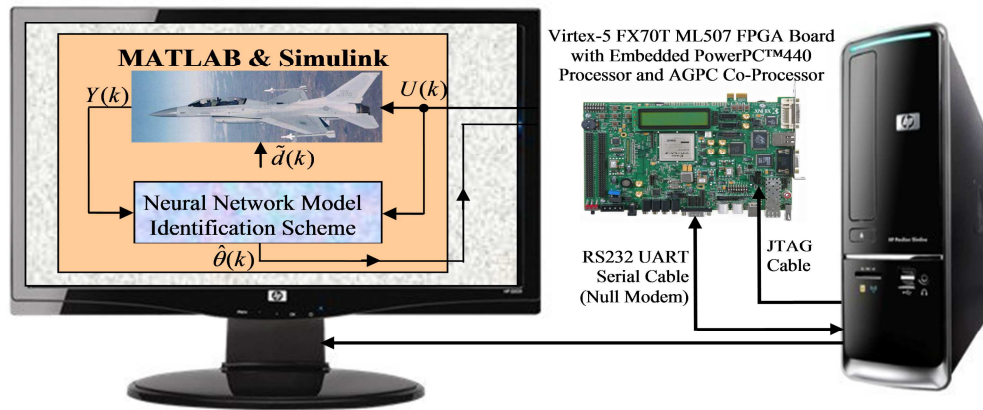


Figure 19. The proposed scheme for the FPGA implementation, verification and performance evaluation of the neural network-based AGPC algorithm on a Xilinx Virtex-5 FX70T ML507 FPGA board.

In Figure 19, the embedded programs that represents the AGPC algorithm, the embedded PowerPC™440 processor, and other memory and hardware device drivers are downloaded through the JTAG cable; whereas the communication between the host computer and the Virtex-5 FX70T ML507 FPGA board is accomplished through the RS232 UART serial cable (null modem). The term “null modem” is used to indicate that the RS232 serial cable transmit and receive lines on the host computer and the Virtex-5 ML507 FPGA board are cross-linked. Unlike the standard RS232 serial cable where it is assumed that one end is data terminal equipment (DTE) and the other end is data circuit-terminating equipment (DCE), the null modem connection is not covered by a standard based on a specific wiring layout. Finally, the connection between the host computer and the display monitor is a VGA cable which delivers the information contents of the host computer for

display. Of the three connections shown in Figure 19, note that only the RS232 serial cable is bi-directional for data transmit and receive operations.

The proposed technique shown in Figure 19 can be summarized as follows:

- 1) It is assumed that neural network (NN) has been previously trained based on experimental data obtained from the simulation of the nonlinear F-16 aircraft model, that the optimal network parameters have been selected as in Section 4.3, and a validated NNARMAX model $\hat{\theta}(k)$ has been obtained;
- 2) It is also assumed that the obtained NNARMAX have been used to simulate the AGPC subject to the constraints given in Table 3 and the optimal AGPC control parameters have been selected as given in Table 4;
- 3) The simulations between the host computer and the Virtex-5 FPGA board are controlled by a file named

“Flag_a” which is resident on the host computer with initial content “a”. The simulations are initialized on the host computer. At the end of the NN model identification process, the identification algorithm writes letter “b” to “Flag_a” to indicate end of identification process. This “b” initializes the embedded AGPC algorithm on the FPGA, which continuously scans “Flag_a” in search for “b” in order to compute the new control inputs. At the end of the control inputs computation, the embedded AGPC scheme writes an “a” back to “Flag_a” to mark end of control inputs computation and initiate a new identification and control sequence. These read/write and receive/transmit are performed via the RS232 serial cables and serial ports on the host computer and the Virtex-5 FX70T ML507 FPGA board. As in Section 4.4, all simulations in this Section are set for $k = 1$ to 160 samples.

- 4) Now, referring Figure 19, at the current sampling instant $k = 1$, the validated Simulink model of the nonlinear F-16 aircraft is simulated with the current input commands in the presence of disturbances $\tilde{d}(k)$ to obtain the output response $Y(k)$. The current inputs and the output response are added to the training data set Z^N and the network is trained to obtain a NN model $\hat{\theta}(k)$ using the MLMA algorithm.
- 5) The obtained NN model $\hat{\theta}(k)$ is then employed for the AGPC controller design to compute the next control inputs that will keep the output response $Y(k)$ close to the desired reference signal $R(k)$. These computations are performed at the current sampling instant $k = 1$ with a time constant $T = 0.5$ and must be completed to update the control inputs within this time to keep the nonlinear F-16 aircraft in its normal route.
- 6) At time $k = k + 1$, the NNARMAX model identification and the AGPC computations are repeated on the basis of the “a” and “b” respectively in the text file “Flag_a”.

5.4. Hardware Synthesis of the AGPC Algorithm Using Xilinx AccelDSP Synthesis Software

The adaptive generalized predictive control (AGPC) algorithm is implemented in this work as MATLAB programs. To realize the hardware implementation of the AGPC algorithm on the Xilinx Virtex-5 FPGA, the AGPC algorithm is first modeled and synthesized using the AccelDSP modeling and synthesis tool [38]. The Xilinx AccelDSP synthesis tool allows for the modeling and synthesis of high-level MATLAB algorithms for realization on Xilinx FPGAs [26–29].

The detailed formulation of the AGPC algorithm as been given in the Section 3.2 and has been used in Section 4.4 for the closed-loop adaptive control of the nonlinear F-16 aircraft. Here, the nonlinear F-16 aircraft NNARMAX model identification and AGPC control objectives as well as the closed-loop adaptive control of the nonlinear AGPC validated with satisfactory results in Section 4.4 based on the desired reference trajectories specified in Figure 1.

So far, the AGPC algorithm has been implemented as MATLAB programs with a script file named “F16_Script.m” and five design function files, namely: “agpc_acceldsp_model.m” as the top-level design function file that calls the AGPC algorithm, “agpc_algorithm.m” which implements the main AGPC algorithm, “ref_filter.m”, which computes and filters the reference signal, “shift.m” which is used to update the predicted outputs and control inputs sequence during the AGPC computations to obtain the optimal control signals, “sigmoid_tanh.m” which is used to extract the input-to-hidden layer weights of the F-16 aircraft NN model.

The objective here is to create a synthesizable hardware model representative of the AGPC algorithm, and so the “Generate System Generator” option is considered here which is shown in the block diagram of Figure 20 [5, 26–29]. The eight procedures required to generate the hardware algorithmic model of the AGPC algorithm are shown in Figure 20.

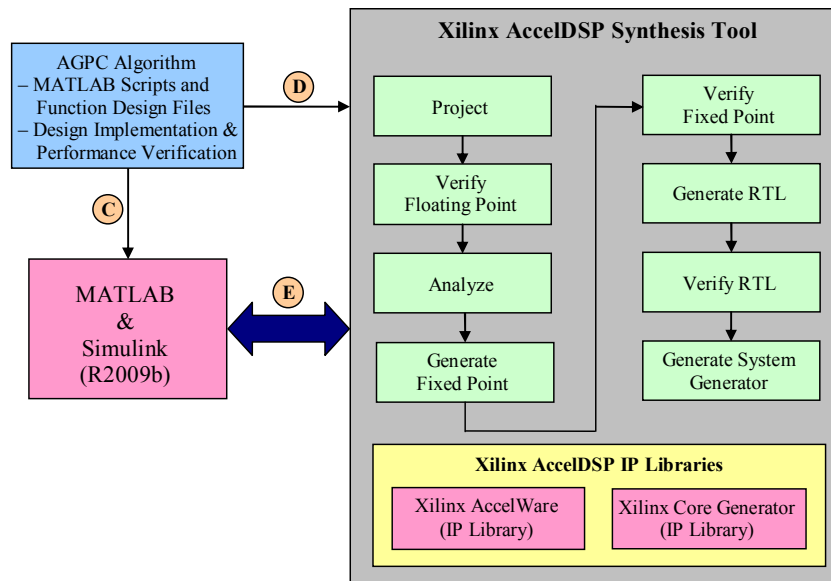


Figure 20. The block diagram of the AGPC modeling and Synthesis using MATLAB and Xilinx AccelDSP synthesis tool.

The procedures in Figure 20 begin with the top-left block. The NNARMAX model identification scheme and the AGPC control scheme are programmed to run in closed-loop with the nonlinear F-16 aircraft model in MATLAB. The script file “*F16_Script.m*” invokes the nonlinear F-16 model identification algorithm which identifies the NNARMAX model of the nonlinear F-16 aircraft and subsequently invokes the top-level design function file “*agpc_acceldsp_model.m*” which implement the AGPC algorithm for $k=160$ samples, subject to the constraints given in Table 3. As discussed Section 5.2, at the end of each of the model identification process “*b*” is written to “*Flag_a*” and the NNARMAX model is also placed in the AGPC directory. Based on “*b*”, the AGPC algorithm uses the new model to compute the new control signals which is used to control the nonlinear F-16 aircraft and write an “*a*” to “*Flag_a*” at the end of the control computations. And the model identification process is repeated for the next time sample and consequently the controller computation and implementation until $k=160$. This online adaptive model identification and control is the verification phase of the AGPC algorithm which are the functions of the two blocks on the left side of Figure 20 and is indicated by the path with circled (C).

The path with circled (E) is the synthesis phase where the pre-verified AGPC algorithmic MATLAB files are

synthesized with the Xilinx AccelDSP synthesis tool. The Xilinx AccelDSP tool also include sets of intellectual property (IP) libraries (Xilinx AccelWare and Xilinx Core Generator functions) which can be included in MATLAB algorithmic programs. As shown on the right side in Figure 20, the Xilinx AccelDSP synthesis tool consists of eight stages to generate the hardware model representative that will encrypt the all the five design function files of the AGPC algorithm. These stages are summarized as follows:

Stage 1) Project: The Xilinx AccelDSP is launched to open the AccelDSP GUI as shown in Figure 21. The project name is specified here as “*aggpc_acceldsp*” and the project directory where all design file, generated files and generated reports will be stored is also specified. The design flow is set to “*System Generator*”; the RTL language is set to “*VHDL*”; the fixed point language is set to “*C++*”; the technology is set to “*Virtex-5*”; the device is “*XC5VFX70T*”; the speed grade is specified as “*-1*”; and the system frequency is set to “*100 MHz*”. These selected parameters are shown in Figure 21. In the FPGA implementation that follows, all “sampling times”, “BLOCK PERIOD” and “BLOCK LATENCY” are made with respect to this system clock frequency which is fixed for all embedded processors designed for Virtex-5 FX70T ML507 FPGA board [5, 26–29].

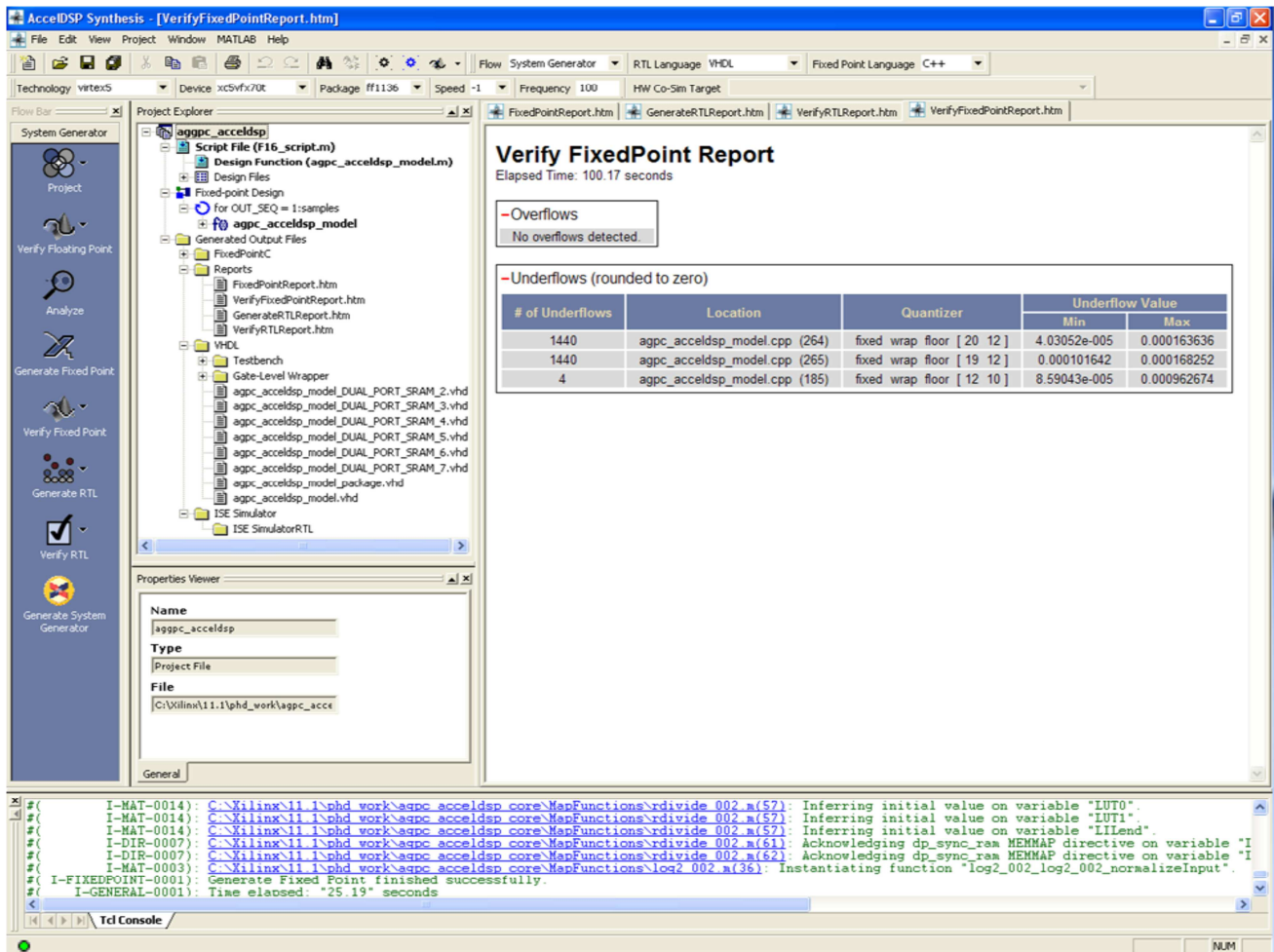


Figure 21. AccelDSP design flow to generate the System Generator block model that encrypts the AGPC algorithm.

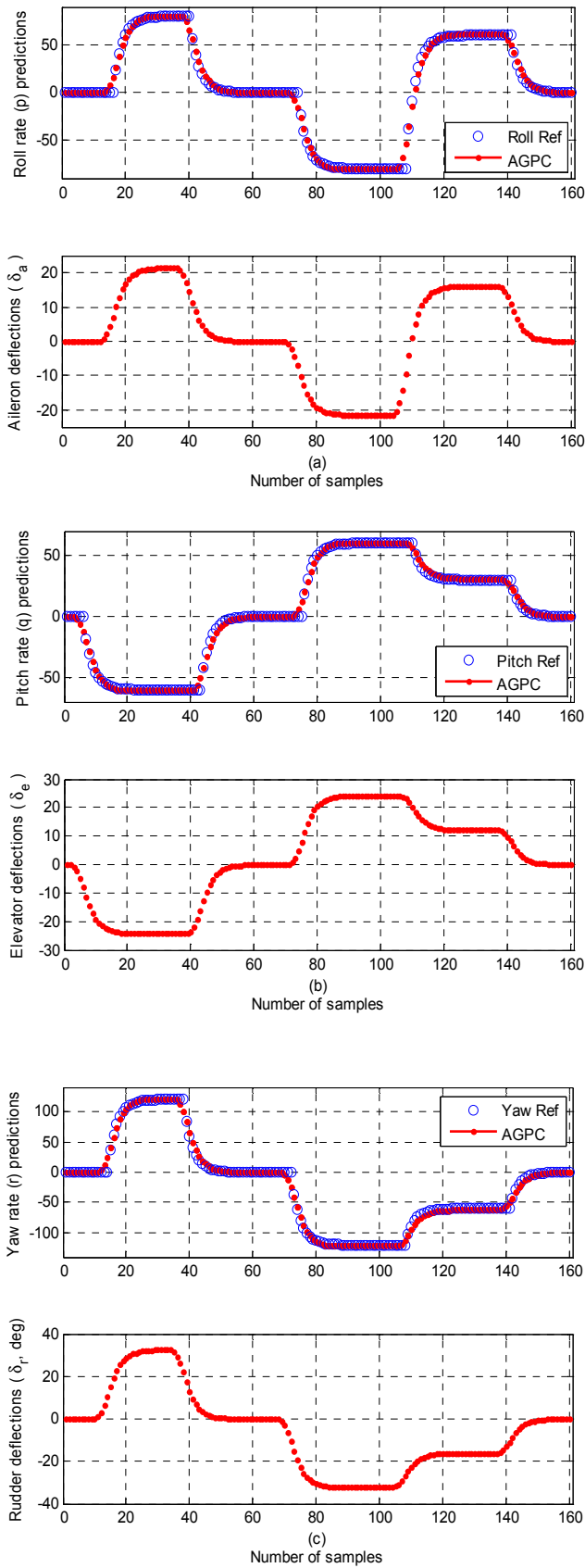


Figure 22. Floating-point simulation results of the F-16 aircraft control using the MATLAB AGPC algorithm with a total computation time of 104.8105 seconds.

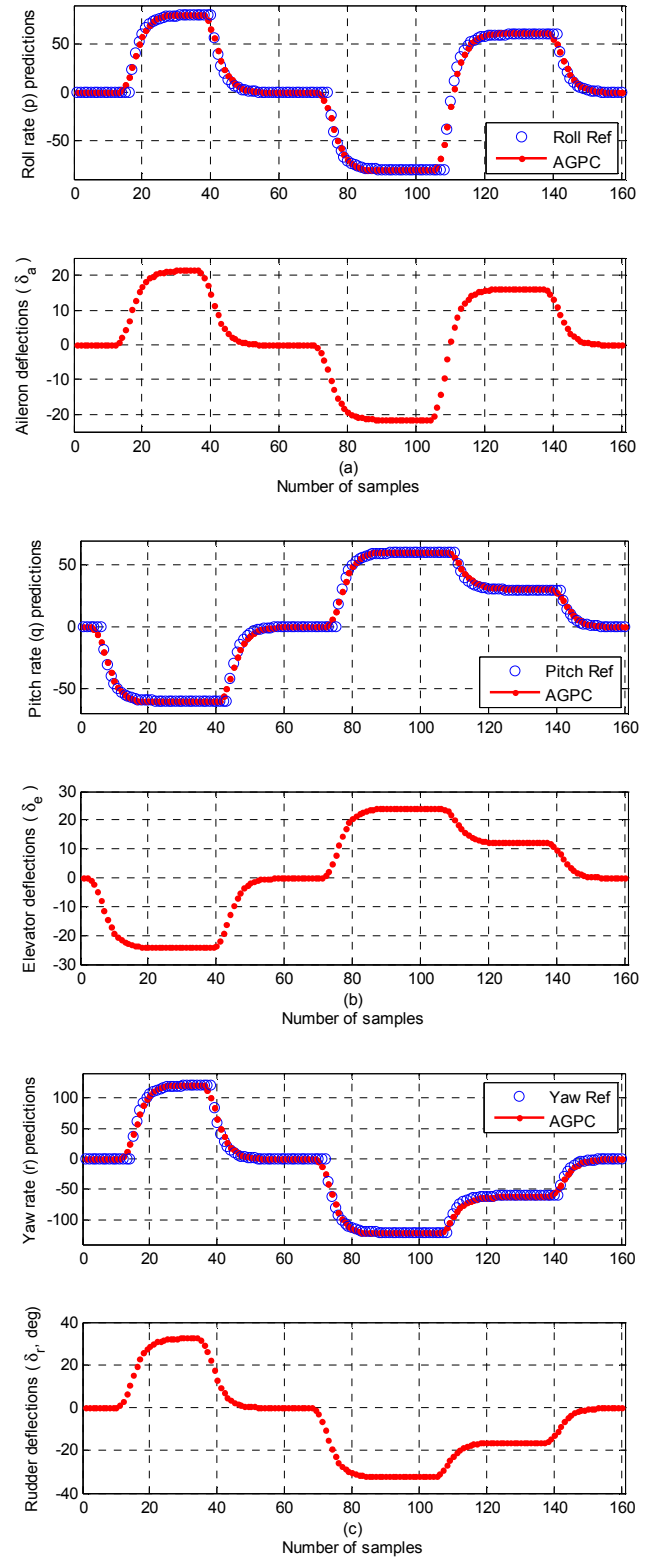


Figure 23. AccelDSP fixed-point simulation of the F-16 aircraft control using the C++ AGPC algorithm with a total computation time 100.17 seconds.

Stage 2) Verify Floating Point: This stage prompts for the script file which in this case is “F16_Script”. The AccelDSP synthesis tool uses this script file to implement and verifies the floating point MATLAB AGPC algorithm. The F-16 aircraft

AGPC control simulation results are shown in Figure 22. As can be seen in Figure 22, the control performance of the AGPC for the nonlinear F-16 aircraft control is acceptable due to the good tracking of the desired reference signals.

Stage 3) Analyze: The stage prompts for the AGPC algorithm top-level function file “*agpc_acceldsp_model*”, and then performs extensive analysis on this file and its sub-function files to ensure that they are fully synthesizable and that they conform to the minimum AccelDSP style guidelines described in the MATLAB for Synthesis Style Guide [40]. Extensive simulations were performed to ensure that the AGPC algorithm conforms to the synthesizable AccelDSP style formats.

Stage 4) Generate Fixed Point: The AccelDSP synthesis tool generates a fixed point equivalent of the floating point MATLAB AGPC algorithm and all the design function files. An intensive quantization of the design was performed at this stage so that the resulting fixed point AGPC algorithm will produce results that are identical or approximately the same as the floating point AGPC algorithm. Since “*Overflows*” have more severe effects on the design than the “*Underflows*”, significant efforts were made to eliminate “*Overflows*” in order to achieve fairly accurate results. For complete details and discussions on *quantization*, *Overflows*, *Underflows* and their effects on “*Generate Fixed Point*” results, the reader is referred to the following references [39, 40].

Stage 5) Verify Fixed Point: The generated fixed point is verified and comparisons are made with the floating point equivalent. If significant discrepancies in the results exist, the quantization process is performed again and the “*Generated Fixed Point*” and the “*Verify Fixed Point*” are both repeated. Stages 3), 4), and 5) required extensive simulations to achieve accurate results. As it is evident in Figure 21, the “*Overflows*” were completely eliminated while three “*Underflows*” could not be eliminated. The “*Verify Fixed Point*” simulation results for the nonlinear F-16 aircraft AGPC control is shown in Figure 23. By comparing the floating point simulation results of Figure 22 and the fixed point results, it can be observed that the AGPC control performance of the fixed point closely follows those obtained by the floating point AGPC algorithm good tracking of the desired reference signals. During the *Verify Fixed Point*, the AccelDSP Synthesis tool captures the data stream on the design inputs and outputs which will be used for “*bit-true*” comparison in later stages of the design.

Stage 6) Generate RTL: This stage generates the register-transfer-level (RTL) model in VHDL from the in-memory design data base. During the “*Generate RTL*” process, all hardware logic based on the design specification and configurations specified during the modification of the design in *Step 3)*, are automatically generated and inferred. The inputs and outputs (I/O) ports for the design are also mapped and implemented. This stage also generates a VHDL “*Testbench.vhd*” file and is stored in the *Testbench* directory shown in Figure 21 that will be used to verify the RTL model. The generated *Testbench* is the VHDL equivalent verification constructs of the MATLAB input and output data stream captured during the *Verify Fixed Point* stage.

Stage 7) Verify RTL: The AccelDSP synthesis tool automatically invoke the Xilinx ISE simulator (ISim) which first compile the VHDL files of the RTL model [26–29]. Next, it verifies the generated VHDL file of the RTL model by applying the generated verification constructs in the *Testbench*, and monitors and compares the output results. The *Verify RTL* stage reports a “*FAILED*” or *PASSED*” depending on the outputs comparison result. A *PASSED* was reported during the *Verify RTL* simulation which implied that the AGPC algorithm has been correctly mapped to its RTL model.

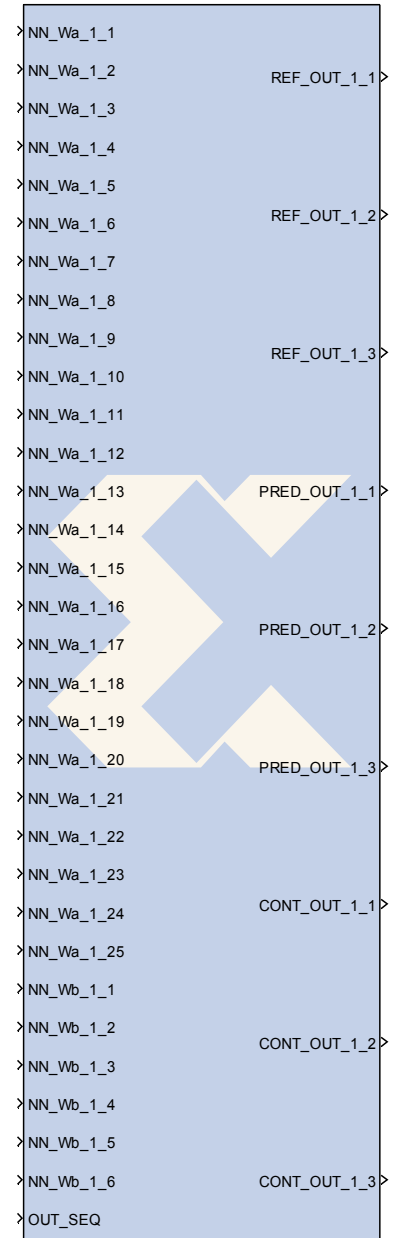


Figure 24. The System Generator block model of the AGPC algorithm generated by Xilinx AccelDSP synthesis tool. Output sequence 1, 2 and 3 corresponds to aileron – roll, elevator – pitch and rudder – yaw signals respectively.

Stage 8) Generate System Generator: This stage generates the synthesized hardware model of the AGPC algorithm

shown in Figure 24. As shown in Figure 24, the inputs to the AGPC hardware model are the NN model parameters as well as the number of samples. The inputs are passed in parallel so that on a single global clock pulse, all the inputs are used in a *frame-based* fashion to produce the outputs.

The generated algorithmic hardware model of the AGPC algorithm will be referred to as “*agpc_accelsdp_model*” which was the name of the top-level function used in creating the algorithmic hardware model. The AccelDSP Synthesis Tool description of the generated hardware model “*agpc_accelsdp_model*” is summarized in Figure 25. The block period is 91 which is the sampling period of the generated hardware model while the latency is 92 by adding one clock cycle to the block period in order to register the inputs. The input-to-hidden layer and hidden-to-output layer weights are designated *NN_Wa_1_X* and *NN_Wb_1_Y* respectively, where *X* is 1 to 25 and *Y* is 1 to 6. The *NN_Wa_1_X* are each of type “*Fix*” with 12 bits for the integer part and 10 bits for the binary part; whereas those for *NN_Wb_1_Y* are type “*Fix*” with 20 bits for the integer part and 12 bits for the binary part. The *OUT_SEQ* is of type “*UFix*” with 8 and 0 bits for the integer and binary parts respectively. The outputs *REF_OUT*, *PRED_OUT* and *CONT_OUT* corresponds to the desired reference signals, predicted outputs and the computed control inputs respectively were all set to type “*Fix*” with 20 and 12 bits for the integer and binary parts respectively. The data types, number of bits and binary points were selected on trial-and-error method (the so-called quantization process); and for each trial *Stage 4) Generate Fixed Point* and *Stage 5) Verify Fixed Point* were repeated until the *Overflows* were eliminated and the simulation results of Figure 23 were obtained.

5.5. Model-Based Implementation of the Synthesized AGPC Algorithm Using System Generator for DSP

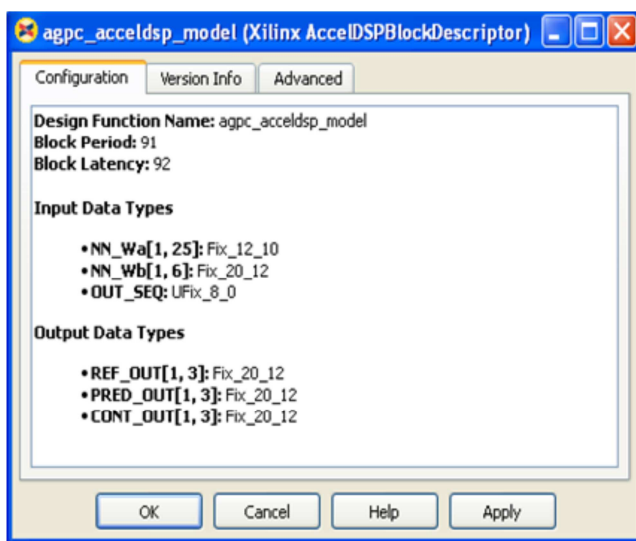


Figure 25. The AccelDSP Synthesis Tool description of the generated hardware model of the AGPC algorithm “*agpc_accelsdp_model*”. The block period corresponds to 91 clock pulses of the FPGA 100-MHz reference clock which is 0.91μs.

The generated hardware model that encrypts the AGPC algorithm shown in Figure 24 is employed in this subsection to build the complete System Generator model of the AGPC algorithm. The input and output interfaces to the *agpc_accelsdp_model* are the shared memories “*From Register*” and “*To Register*” blocks which are taken from the System Generator library block in the Simulink library browser. The complete System Generator model of the AGPC algorithm is shown in Figure 26. These shared memories are used so that an addressable memory mapped interface can be created which can be used to write to and read from the *agpc_accelsdp_model*.

The *agpc_accelsdp_model* receives the neural network weights via a bank of 31 shared memory “*To Register*” blocks. As shown in Figure 26, the input-to-hidden layer weights are grouped as sub-system into the block *Wa_Input_Regs* while the hidden-to-output weights are grouped into the block *Wb_Input_Regs*. The initialization and implementation of the complete System Generator model for the AGPC algorithm of Figure 26 is controlled by the “*Flag_a*” via the input “*c*” as discussed previously. If “*c* = *a*”, the model identification process is implemented. On completion, the model identification algorithm writes a “*b*” to the file “*Flag_a*”, which is used to initialize and implements the AGPC algorithm of Figure 26. On completion, the AGPC algorithm writes an “*a*” which is used to repeat the model identification process for the number of samples, $k = 14560$ samples. The number of samples is specified via a “*Counter Limiter*” as 14560 samples. Then, the Xilinx Gateway In block, named as “*IN_OUT_SEQ*” in Figure 26, is used to convert the Simulink integer, double and fixed point data type from the *Counter Limiter* into System Generator fixed point data type.

The outputs of the “*agpc_accelsdp_model*” are connected to nine shared memory “*From Register*” blocks, namely: *AIL_REF*, *ELEV_REF*, *RUDD_REF* for the reference signals; *AIL_PRED*, *ELEV_PRED*, *RUDD_PRED* for the predicted outputs; and *AIL_ROLL_CONT*, *ELEV_PITCH_CONT*, *RUDD_YAW_CONT* for the control signals. Where *AIL*, *ELEV* and *RUDD* represents aileron, elevator and rudder respectively. Again, these shared memories are used so that an addressable memory mapped interface can be created through which these registers can be accessed for a write operation by the “*agpc_accelsdp_model*” and read operation by a peripheral. For the purpose of evaluating the performance of the complete AGPC algorithm of Figure 26 in Simulink, these outputs registers are connected to the Xilinx “*Gateway Out*” blocks shown in Figure 26. These Gateway Out block converts the System Generation fixed point data types to Simulink integer, double, fixed point data types for plotting on the Simulink scope blocks.

As a general rule, every System Generator design must include at least the System Generator Token at the top-level in the highest hierarchy of the design. The System Generator Token is included and shown at the bottom of Figure 26 [5, 26-29]. Note that the block is not connected to any other block, rather it act as an interface to the Xilinx design and simulation tools [5].

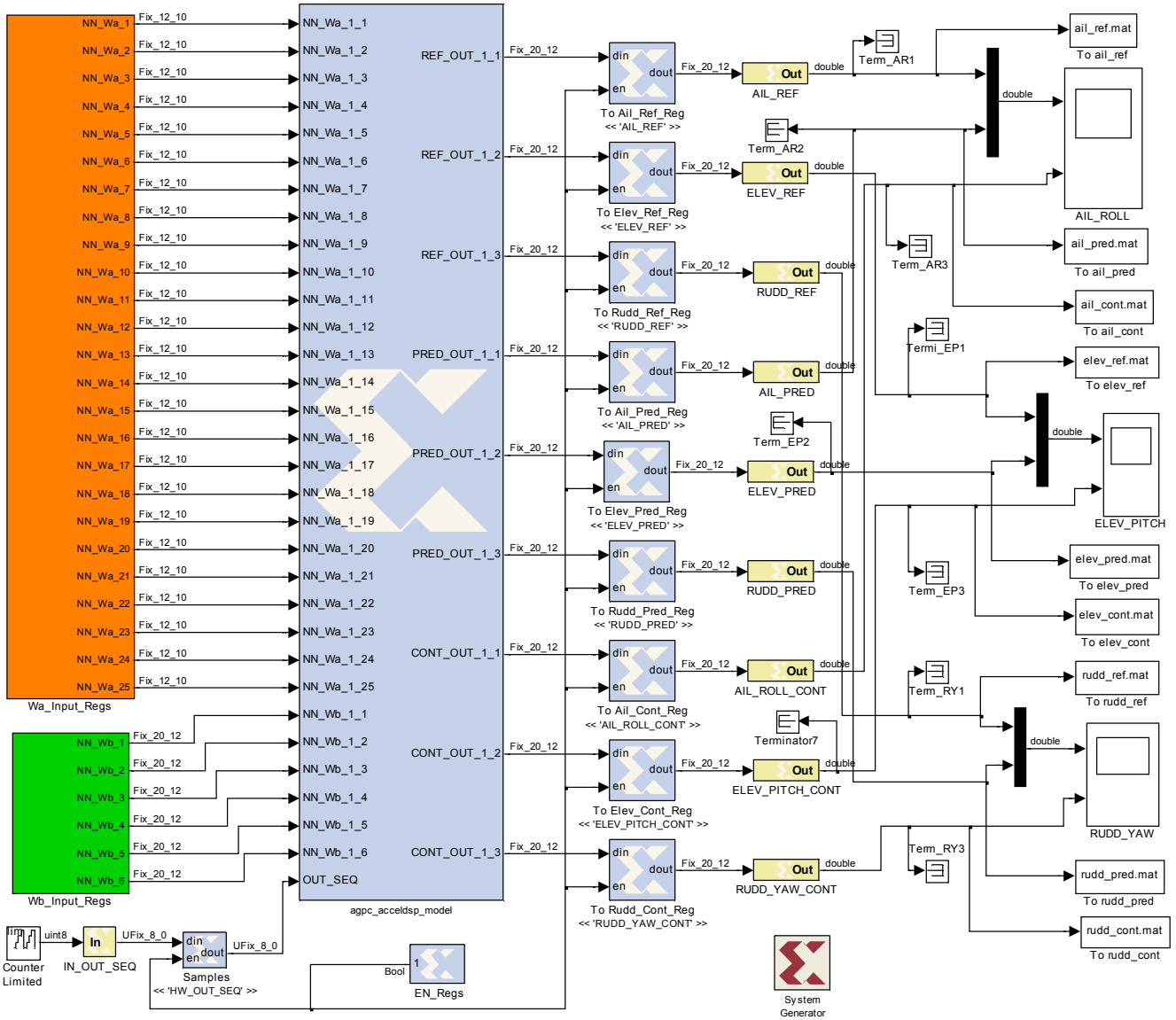


Figure 26. The complete System Generator model for the generated hardware model “*agpc_acceldsp_model*” for the AGPC algorithm.

Next, the complete System Generator model of Figure 26 is implemented in closed-loop with the nonlinear F-16 aircraft Simulink model and neural network model identification scheme using the ARLS algorithm. As discussed earlier, due to the block period of 91 clock pulses with respect to the 100-MHz reference clock frequency, the number of Simulink simulations is specified as 14560 samples. The closed-loop nonlinear F-16 aircraft identification and control simulation results are shown in Figure 27. It can be observed that the System Generator model of the AGPC algorithm closely track the desired reference signals.

Although, by comparing Figure 27 with those of Figure 22 and Figure 23, a small mismatch appears in the fourth control sample sequence in the two graphs of Figure 27(a) while another can also be observed in the top figures of Figure 27(b) and (c). However, the model identification and control performance of the complete System Generator model of the AGPC algorithm using “*agpc_acceldsp_model*” gives acceptable control results. The

computation time for the nonlinear F-16 aircraft control for 160 samples at the “*agpc_acceldsp_model*” block period of 91 Reference Clock Frequency of 100MHz over the 14560 samples using the complete System Generator model of the AGPC algorithm is 1.8815 seconds. This implies that each control action is executed in 0.12922 *ms* (milliseconds). This gives an improvement of 5.0689×10^3 times faster when compared to the computation time obtained with MATLAB floating-point AGPC algorithm.

Although, the achieved computation time of 0.12922 *ms* by the System Generator mode of the AGPC algorithm is approximately 3.8694×10^3 times below the 0.5 seconds time constant of the nonlinear F-16 aircraft [1] based on the AGPC control simulations only, additional time will be introduced by the model identification scheme as well as actuator time constants not covered here. Hence, additional improvements, if possible, are necessary for further computation time reduction. One approach for this improvement is to integrate the System

Generator model of the AGPC algorithm with an embedded processor system as a co-processing hardware. By this way, the synthesized AGPC algorithm can be executed at or close to the embedded processor system's operating frequency.

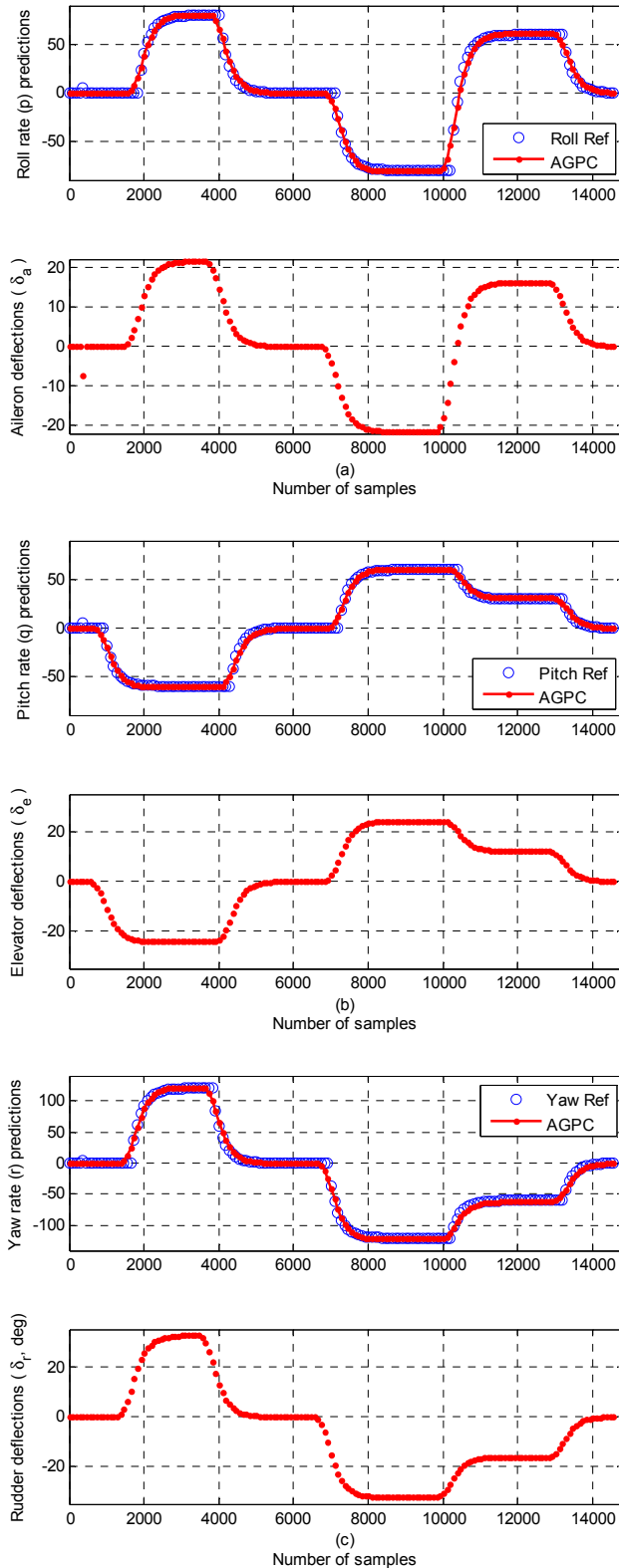


Figure 27. The nonlinear F-16 aircraft control simulation results using the System Generator model of the AGPC algorithm of Figure 26.

Before integrating the System Generator model of the AGPC algorithm with an embedded processor system, it is necessary to test the closed-loop performance of this model in a FPGA-in-the-loop hardware co-simulation with the Virtex-5 ML507 FPGA board.

5.6. Hardware-in-the-Loop Co-Simulation of the System Generator Model of the Synthesized AGPC Algorithm on Xilinx Virtex-5 FX70T ML507 FPGA Board

System Generator for DSP provides hardware-in-the-loop (HIL) co-simulation (HW Co-Sim) which makes it possible to incorporate the System Generator model for the AGPC algorithm running in Simulink directly into the Virtex-5 FX70T ML507 FPGA board. This allows the compiled portion of the System Generator model to be tested in the actual Virtex-5 FX70T ML507 FPGA board. Through the HW Co-Sim, the performance of the System Generator model of the AGPC algorithm when downloaded to the Virtex-5 FX70T ML507 FPGA board can be verified in advance, and modification through all the phases of the design can be made to correct errors.

From the point of view of model-based system design and development, the Xilinx System Generator for digital signal processing (DSP) enables the use of Simulink-MATLAB modeling and simulation environment for FPGA design by providing a smooth path from initial design capture via the System Generator token (shown on the left-hand side of Figure 28) to Xilinx FPGA implementation and analysis.

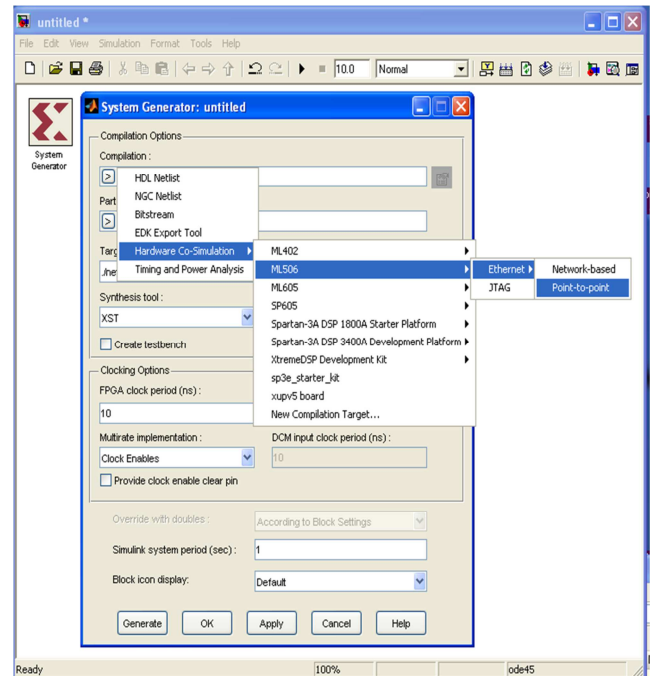


Figure 28. System Generator token (left) and the six System Generator compilation options (right) with available Hardware Co-Simulation options without the Virtex-5 ML507 FPGA board.

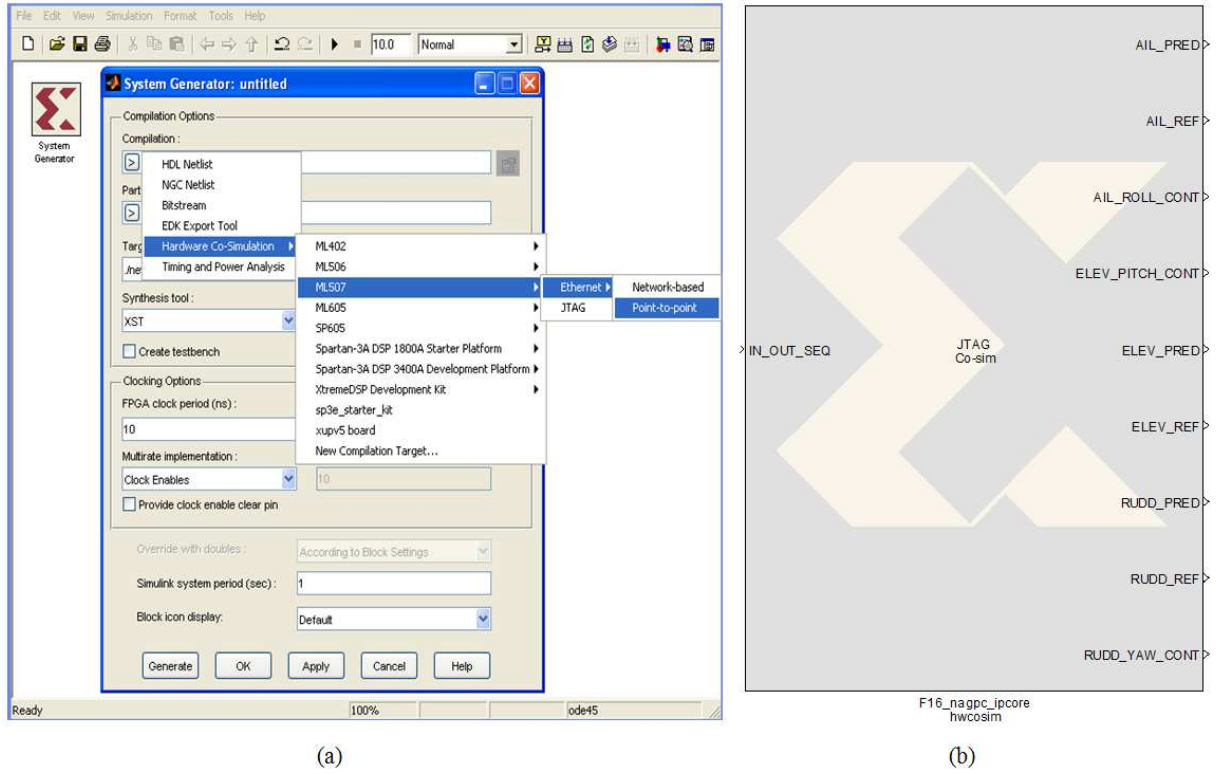


Figure 29. (a) System Generator token (left) and the six System Generator compilation options with the Hardware Co-Simulation options for Virtex-5 ML507 (right) and (b) Hardware Co-Simulation block.

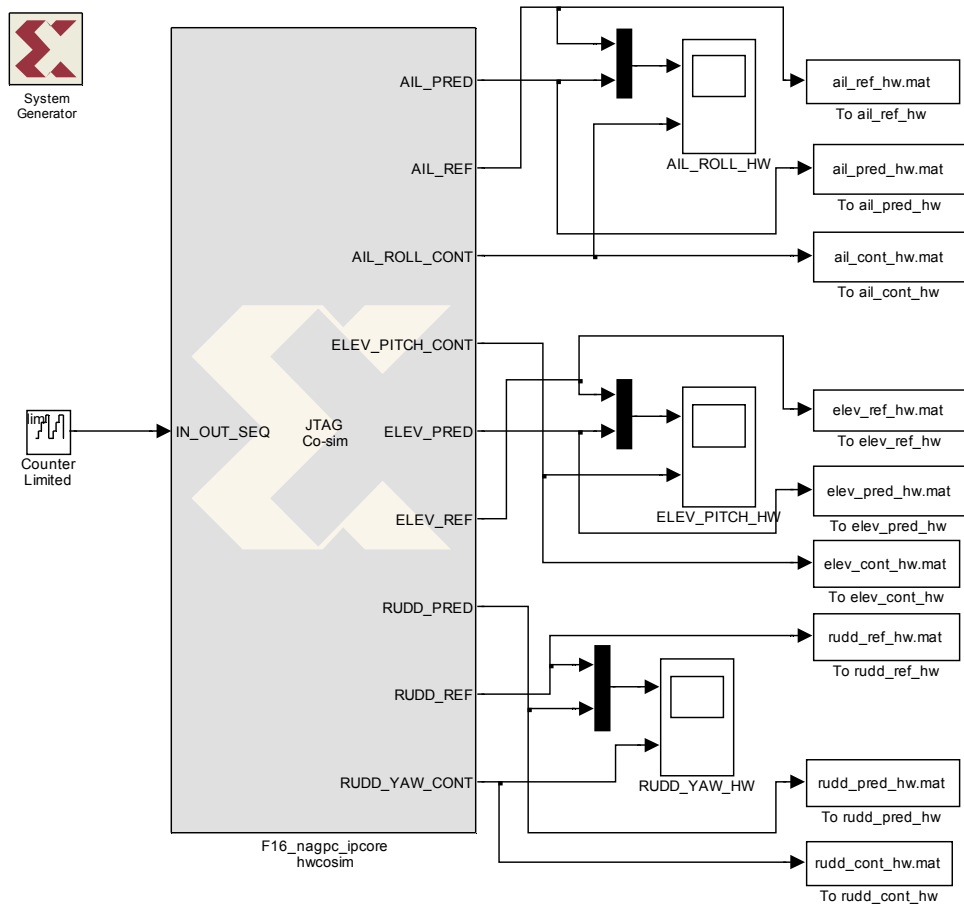


Figure 30. The System Generator model of the AGPC algorithm for the nonlinear F-16 aircraft auto-pilot control with the generated Hardware Co-Simulation block.

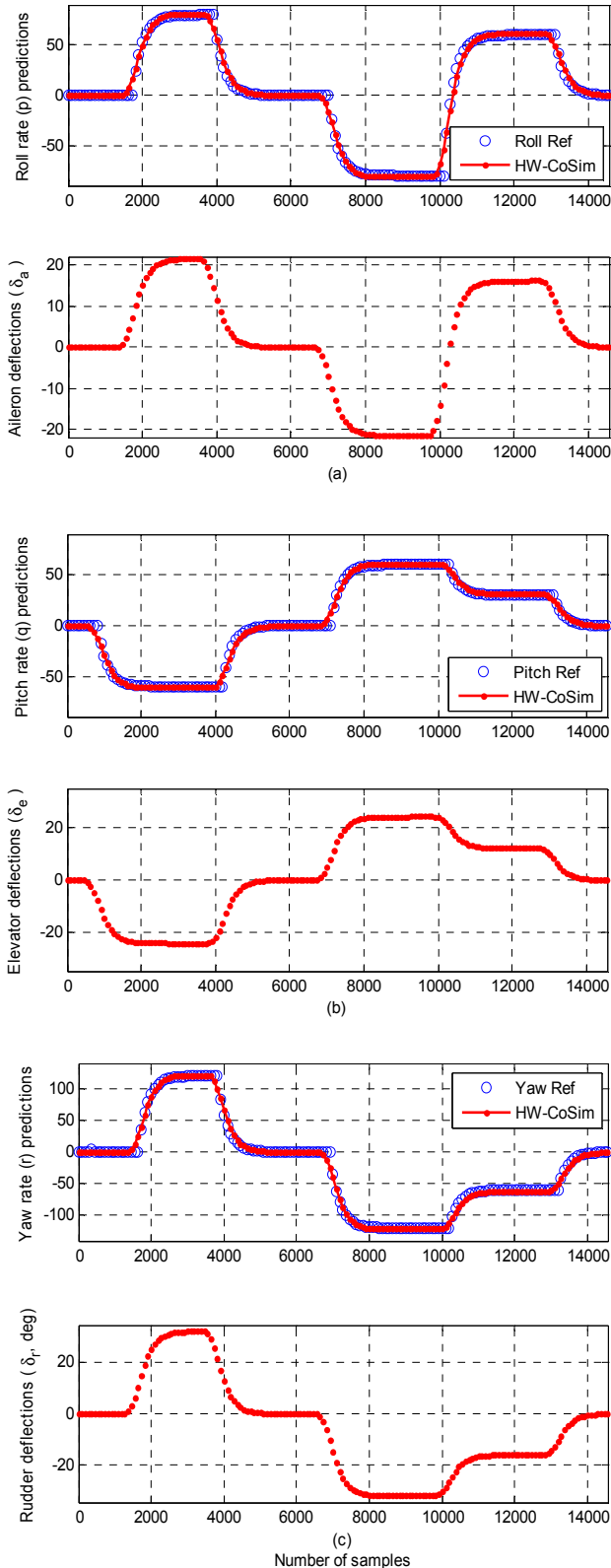


Figure 31. Hardware-in-the-loop co-simulation results produced by the generated Hardware Co-Simulation block model evaluated on the Xilinx Virtex-5 ML507 FPGA board over JTAG cable. In the top plots, the output predictions (yellow) are compared to the reference signal (red). The bottom plots are the control signals. (a), (b), and (c) are the simulation results for the aileron-roll, elevator-pitch and rudder-yaw prediction and control respectively.

In order to perform this HW Co-Sim of the System Generator model for the AGPC algorithm on Virtex-5 FX70T ML507 FPGA board, the HW Co-Sim block must first be generated using the System Generator Token. As Figure 28 shows, the Virtex-5 FX70T ML507 FPGA board support for HW Co-Sim is not available. Here, all the four MATLAB program files that configure the ML506 FPGA board for HW Co-Sim are copied to a new sub-directory and modified for the Virtex-5 FX70T ML507 FPGA board. The four files are *ML506_JTAG.ucf*, *ML506_JTAG_PostGeneration.m*, *ML506_JTAG_Target.m* and *xtarget.m*. The modifications are made by changing all the “6” to “7” as well as the specifications of the Virtex-5 XC5VFX70T as shown in Figure 21 for the ML507 board. These modifications now allow the Virtex-5 FX70T ML507 FPGA board to be used for HW Co-Sim via the System Generator Token as shown in Figure 29(a) when compared to Figure 28.

Now, using the System Generator model of the AGPC algorithm of Figure 26 and the System Generator Token of Figure 29(a), the HW Co-Sim block for the System Generator model is generated which is shown in Figure 29(b). As can be seen in Figure 29(a), both Ethernet and JTAG options for the HW Co-Sim is available for use, the JTAG interface option is used in this work. The reason is that the Ethernet HW Co-Sim interface utilizes an evaluation version of the Ethernet MAC IP core which becomes dysfunctional after continuous, prolonged operation in the FPGA board for approximately 7 hours.

To set up the JTAG HW Co-Sim simulation model, the “Counter Limiter” and the “Scope” blocks from Simulink libraries, similar to that in Figure 26, are added to the generated JTAG HW Co-Sim block of Figure 29(b) to obtain the complete HW Co-Sim simulation model shown in Figure 30. Next, the Virtex-5 FX70T ML507 FPGA development board is connected similar to the scheme shown in Figure 19. The closed-loop hardware-in-the-loop co-simulation with the Virtex-5 FX70T ML507 FPGA board is performed.

The control simulation results are shown in Figure 31. Comparing the control results of Figure 31 to those of Figure 22, Figure 23 and Figure 27; it can be observed that the JTAG HW Co-Sim gives similar good and acceptable control performances. This is an indication that the design System Generator model of the AGPC algorithm is a good representative of the original floating pointing MATLAB algorithm. This also justify that a good performance will be achieved from the System Generator model of the AGPC algorithm when it is programmed into the Virtex-5 FX70T ML507 FPGA.

The computation time for the nonlinear F-16 aircraft control for 14560 samples with the “*agpc_acceldsp_model*” using the JTAG HW Co-Sim block generated from the System Generator model of the AGPC algorithm is 9.2561 seconds. This implies that AGPC control action is executed in 0.63572 ms at each time sample. Although this give an improvement of 1.0303×10^3 times faster over the original floating point MATLAB algorithm. Also, the achieved computation time of

0.63572 ms by the HW Co-Sim block model is approximately 7.8651×10^2 times below the 0.5 seconds time constant of the nonlinear F-16 aircraft [1] based on the control simulations only. The comparing the computation time of 0.63572 ms obtained with the HW Co-Sim block model with the 0.12922 ms obtained with System Generator model, it is obvious that the HW Co-Sim block model has increased computation time of 4.9197 seconds.

Thus, the System Generator model implementation is about 5 times faster than the FPGA-in-the-loop implementation of the synthesized AGPC algorithmic model than the HW Co-Sim block model. This increase in computation time is not unusual [41].

The possible reasons for increased computation time are: 1) because the HW Co-Sim block is in effect producing the clock signal for the Virtex-5 FPGA board only when Simulink initializes it, 2) the overhead associated with the rest of the Simulink model's simulation, and 3) the communication overhead due to bus latency between Simulink and the Virtex-5 FPGA can significantly limit the performance achieved including increased computation time. In addition to the *Gate-Level wrapper* generated for “*agpc_acceldsp_model*” by the AccelDSP synthesis tool (see Figure 21), System Generator for DSP also generate the memory-map interfaces for the “*agpc_acceldsp_model*”, and provides the necessary hardware interfaces and software drivers for the System Generator model. A brief discussion on HW Co-Sim and a more complete detail on the HW Co-Sim block generation processes can be found in [40, 41].

6. Discussion of Results

6.1. Discussions on the Generated Hardware Model of the AGPC Algorithm

The “Fix” and “UFix” are signed and unsigned fixed point modes. The “Fix” is used to provide greater range for the positive and negative values of the NN weight as well as the outputs which are usually unknown in nature. The “UFix” is used to provide greater range since the number of sample will always be positive numbers. An *Overflow* occurs when the magnitude of a number assigned to a variable exceeds the number of bits allocated to the integer part of the fixed-point word. On the other hand, an *Underflow* occurs when a very small fractional number gets rounded to zero. *Underflows* are usually more common and less serious than *Overflows* [40, 41]. For example, the observed minimum values of the NN input-to-hidden layer weights were in the order of 10^{-5} , and this posed more challenges on quantizing their fixed point equivalent.

The amount of error between the floating and the fixed point design is called quantization error. Because AccelDSP Auto-Quantizer recognizes the MATLAB 53-bit limit for simulating bit-true fixed point arithmetic, the maximum “Fix” and “UFix” bits are limited to these value in this implementation. Additional, challenge is to ensure that no addition, division or multiplication of two variables in the

AGPC algorithm resulted in value greater than 53-bits. As the resulting hardware cost increases with the number of bits, and the numbers of bits for the integer part must be greater than the numbers of bits for the binary parts; significant efforts have been made to keep the number of bits as small as possible while ensuring the accuracy of the resulting fixed-point algorithm without *Overflows*.

Besides the hardware cost but as shown in Table 5, the maximum number of input-output (I/O) ports available for use on the Virtex-5 FX70T ML507 FPGA embedded system development board is 640. Here, the maximum input-output (I/O) ports used by the generated hardware model are 613 out of the 640 ports [5]. In addition to the 608 I/O ports used which can be calculated from Figure 25, five additional “UFix” data type ports were created with number of bits 1 and binary point 0 respectively. Four of these ports are input ports while the fifth is an output port, which listed consecutively as follows:

- 1) *Clock*: The generated hardware model for the AGPC algorithm has one global clock input. Data transfer on each data ports are synchronized to the clock. The clock frequency corresponds to the 100-MHz specified in Figure 21.
- 2) *ClockEnable*: The ClockEnable enables the clock.
- 3) *Reset*: The global reset must be held active high for at least one clock cycle and returns all registers to known state. The “*Generate System Generator*” option for the AccelDSP synthesis flow used in this work (see Figure 20 and Figure 21) processes the all data at fixed rate and has a constant throughput. Constant throughput means that all required tasks must be completed within the specified block sample period. This was verified by the “*PASSED*” issued by the *Verify RTL* stage of the hardware synthesis.

When the “*Generate System Generator*” option is specified, it is expected that the generated hardware model will be part of a larger System Generator design, as it is in this work. Therefore, the generated hardware model must process and deliver data placed at its input ports to the output pots within the block sample period. This input-output communication is controlled by an interface protocol, namely: the full handshake protocol and the push-mode handshake protocol. Unlike the full handshake protocol supported by the *ISE* option (that is the *Verify Gate Level* in Figure 20) which can be used when the design does not have constant throughput, the *Generate System Generator* option implements the push-mode handshake protocol. Unlike the full handshake protocol, the push-mode handshake protocol does not rely on input request from the generated hardware before data is sent and output acknowledgement that data has been received by the output device. This is because the push-mode protocol is limited to designs with a constant throughput [40, 41].

Note that if the AccelDSP is unable to the push-mode protocol, the full handshake is implemented and the AccelDSP design flow switches to *Verify Gate Level* option. Experiences have shown that the resulting *Gate level* design

produce errors when used in the Xilinx ISE Foundation to generate the programming file. Thus, the MATLAB programs must be modified and all the AccelDSP flow repeated.

- 4) *ac_InputAvail*: The *ac_InputAvail* is simply the input availability. A question here may be from where and how the input is available, and what is the output device? Let the input and output devices be the Xilinx “From Register” and “To Register” blocks respectively [5, 26–29]. The data from the input register is presented at the

output and the hardware model captures and processes the data; and then writes the results to the output register. Latency may then be defined here as the time between when the data is presented and when it is received. Hence, a latency of 1 clock is added in order to compensate for this time lag.

- 5) *ac_OutputAvail*: Also *ac_OutputAvail* is simply the output availability. The processed data from the hardware model is written to the output register. The output register also has a latency of 1 clock cycle.

Table 5. Comparison of the Xilinx General-Purpose, Defense-Grade, Space-Grade Virtex-4 and Virtex-5 FPGA Product Family Members in terms of their available hardware resources and capabilities.

Available Hardware Resources		Virtex-4 FPGA Family Members ⁽¹⁷⁾						Virtex-5 FPGA Family Members ⁽¹⁷⁾				
		XC4V FX12	XC4V FX20	XC4V FX40	XC4V FX60 ⁽¹³⁾	XC4V FX100	XC4V FX140 ⁽¹³⁾	XC5V FX30T ⁽¹⁵⁾	XC5V FX70T	XC5V FX100T	XC5V FX130T ⁽¹⁶⁾	XC5V FX200T
Configurable Logic Blocks (CLBs) ⁽¹⁾	Array ⁽²⁾ [Rows x Cols]	64x24	64x36	96x52	128x52	160x68	192x84	80x38	160x38	160x56	200x56	240x68
	Logic Cells	12,312	19,224	41,904	56,880	94,896	142,128	NA	NA	NA	NA	NA
XtremeDSP Slices ⁽⁴⁾	Slices	5,472	8,544	18,624	25,280	42,176	63,168	NA	NA	NA	NA	NA
	Virtex-5 Slices ⁽³⁾	NA	NA	NA	NA	NA	NA	5,120	11,200	16,000	20,480	30,720
DSP48E Slices ⁽⁵⁾	Max Distributed RAM (Kb)	86	134	291	395	659	987	380	820	1,240	1,580	2,280
	Slices	32	32	48	128	160	192	NA	NA	NA	NA	NA
Block RAM Blocks	18Kb Blocks ⁽⁶⁾	36	68	144	232	376	552	136	296	456	596	912
	36Kb Blocks	NA	NA	NA	NA	NA	NA	68	148	228	298	456
Digital Clock Managers (DCMs)	MAX Block RAM (Kb)	648	1,224	2,592	4,176	6,768	9,936	2,448	5,328	8,208	10,728	16,416
	Phase-Matched Clock Dividers (PMCDs)	4	4	8	12	12	20	NA	NA	NA	NA	NA
Clock Management Tiles (CMTs) ⁽⁷⁾	Clock Management Tiles (CMTs) ⁽⁷⁾	0	0	4	8	8	8	NA	NA	NA	NA	NA
	PowerPC Processors Blocks ⁽⁸⁾	NA	NA	NA	NA	NA	NA	2	6	6	6	6
Endpoint Blocks for PCI Express	PowerPC Processors Blocks ⁽⁸⁾	1	1	2	2	2	2	1	1	2	2	2
	Ethernet MACs ⁽⁹⁾	NA	NA	NA	NA	NA	NA	1	3	3	3	4
Max RocketIO MGT Transceivers Blocks ⁽¹⁰⁾	Ethernet MACs ⁽⁹⁾	2	2	4	4	4	4	4	4	4	6	8
	Max RocketIO MGT Transceivers	NA	8	12	16	20	24	NA	NA	NA	NA	NA
Total Input/Output (I/O) Blocks ⁽¹¹⁾	Transceivers GTX	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
	Blocks ⁽¹⁰⁾	NA	NA	NA	NA	NA	NA	8	16	16	20	24
Max User Input/Output (I/O) ^{(12), (14)}	Total Input/Output (I/O) Blocks ⁽¹¹⁾	9	9	11	13	15	17	12	19	20	24	27
	Max User Input/Output (I/O) ^{(12), (14)}	320	320	448	576	768	896	360	640	680	840	960

Notes on the superscripts indicated in Table 5:

- (1) In the Virtex-4 FPGAs one CLB block contains four slices with 64-bits maximum.
- (2) In the Virtex-4 FPGAs, some of the row/column array is used by the processors in the FX devices.
- (3) Virtex-5 slices are organized differently from previous generations. Each Virtex-5 FPGA slice contains four look-up tables (LUTs) and four flip-flops whereas previous generation of FPGAs had two LUTs and two flip-flops.
- (4) Each XtremeDSP slice contains one 18x18 multiplier, an adder, and an accumulator.
- (5) Each DSP48E slice in the Virtex-5 FPGAs contains a 25x18 multiplier, an adder, and an accumulator.
- (6) Block RAMs are functionally 36-Kbits in size. Each block can also be used as independent 18-Kbit blocks.
- (7) Each Clock Management Tile (CMT) contains two Digital Clock Managers (DCMs) and one phase-lock-loop (PLL).
- (8) PowerPC 405 processor Block(s) are embedded in the Virtex-4, Virtex-4Q and Virtex-4QV FX FPGA members while the enhanced PowerPC 440 processor Block(s) are embedded in the Virtex-5, Virtex-5Q and Virtex-5QV FXT FPGA members.
- (9) The Virtex-5 FPGA family members contain separate Ethernet Memory Access Controllers (Ethernet MACs) per device.
- (10) RocketIO GTP transceivers in the Virtex-5 FPGAs are designed to run from 100-Mbits/s to 3.75-Gbits/s. RocketIO GTX transceivers are to run from 150-Mbits/s to 6.5-Gbits/s whereas those in the Virtex-4 FPGAs are designed to run from 622-Mbits/s to 6.5-Gbits/s only.
- (11) This total number of the input/output (I/O) Banks includes configuration Bank 0.
- (12) The Maximum User Input/Output (Max User I/O) ports do not include the RocketIO transceivers I/O ports.
- (13) Only the Virtex-4 XC4VFX60 and XC4VFX140 counterparts are available in the Virtex-4Q and Virtex-4QV FPGAs; i.e. XQ4VFX60 and XQ4VFX140 as well as XQR4VFX60 and XQR4VFX140 respectively for the Virtex-4Q and Virtex-4QV.
- (14) The Virtex-4 XC4VFX60 has 896 Max User I/Os ports respectively whereas each of the Virtex-4Q and Virtex-4QV counterparts has 768 Max User I/Os ports.
- (15) Of the five designations of the Virtex-5 FPGA members, only the Virtex-5 XC5VFX30T FPGA counterpart; that is Virtex-5 XQ5VFX30T, is not available in the Virtex-5Q FPGA members.
- (16) Of the five designations of the Virtex-5 FPGA members, only the Virtex-5 XC5VFX130T FPGA counterpart; that is Virtex-5 XQR5VFX130T, is available in the Virtex-5QV FPGA members with 836 Max User I/Os ports against 840 in XC5VFX130T. The Virtex-5QV XQR5VFX130T FPGA member has 18 Max RocketIO GTX Transceivers designed to run from 150-Mbits/s to 3.75-Mbits/s only against 20 in XC5VFX130T designed to run in the two ranges given in (10) above. The Virtex-5QV XQR5VFX130T FPGA member does not include the any embedded PowerPC Processor Block(s).
- (17) Apart from the features listed above, the general-purpose, Defense-grade, and the Space-grade Virtex-4 and Virtex-5 series of FPGA family members have approximately the same hardware resources as in Table 5 except for their operational environments.

6.2. Remarks on the Generated Hardware Model of the AGPC Algorithm

By comparing the nonlinear F-16 control simulation results of Figure 22 and Figure 23, it can be observed that the fixed point C++ program of the AGPC algorithm closely follow the floating point MATLAB due to the high accuracy quantization of the floating point algorithm. It can also be observed that C++ program took 100.17 seconds as shown by the *Verify FixedPoint Report* in Figure 21 against the 104.8015 seconds used by the floating point algorithm, which indicates that the fixed point is 4.6315 second faster than the floating point counterpart.

By dividing these computation times by the total number of samples being 160, then the average time required at each sampling instant to compute the control inputs are 0.6550 and 0.6261 seconds (see Figure 32) for the floating and fixed point AGPC algorithms respectively. Note that time excludes that for the neural network model identification. The sampling time of the generated hardware model of the AGPC algorithm in Figure 25 shows the block period to be 91 based on the specified frequency of 100-MHz for the Virtex-5 XC5VFX70T FPGA.

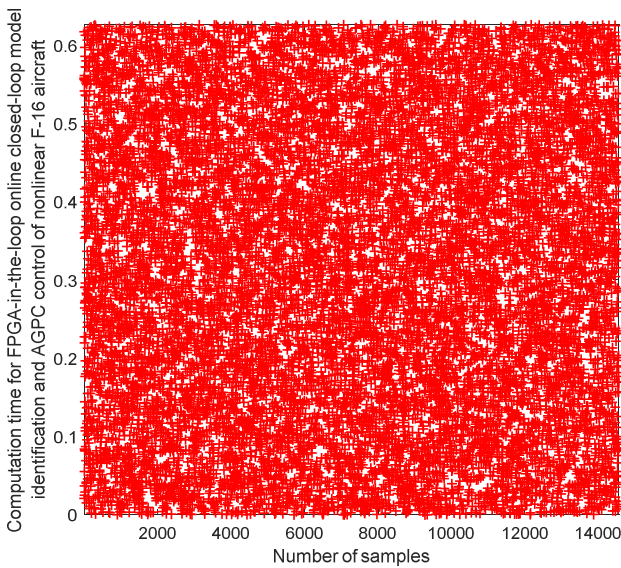


Figure 32. Computation time for the FPGA-in-the-loop implementation of the NNARMAX model identification and AGPC control closed-loop simulation of the nonlinear F-16 auto-pilot control system at each time sample per 91 clock pulses.

The block period of 91 implies that the “*agpc_acceldsp_model*” block model produces an output for a given input sample after 91 clock cycles. The block period is the time it takes the “*agpc_acceldsp_model*” block model to process data presented at its input to the time it produces the results. Thus, given the neural network model, the “*agpc_acceldsp_model*” block model implements the AGPC algorithm at a frequency of 1.0989MHz which is 0.91 μ s (microseconds) at each time sample. Note that since the generated hardware model implements the AGPC algorithm at 0.91 μ s with 91 clock pulses and the expected number of samples is 160, then the total number of simulation

samples is 14560 samples as implemented by the generated hardware model.

7. Conclusion and Further Work

This paper has presented a new comprehensive model-based approach for the online closed-loop neural network-based model identification and adaptive model predictive control strategies. A new step-by-step technique for the modeling, synthesis, mapping, verification, and FPGA-in-the-loop hardware co-simulation has also been presented and validated with satisfactory results. The closed-loop neural network-based model identification and adaptive control strategies can dynamically adapt to changes in time-varying nonlinear systems under different operating conditions.

Note that online closed-loop NNARMAX model identification and AGPC control of the nonlinear F-16 aircraft implemented on an Intel® Core™2 CPU running at 1.86GHz to 6.1048. However, the FPGA-in-the-loop hardware co-simulation of the online closed-loop NNARMAX model identification and AGPC control of the nonlinear F-16 aircraft has shown significant reduction in the computation between the floating-point MATLAB AGPC and fixed-point C++ AGPC algorithms from 0.6550 and 6.1048 to 0.6261 seconds respectively. Even though, the computation time of the synthesized AGPC algorithmic model at each sampling time of 0.6261 is higher than the sampling time of the nonlinear F-16 aircraft of 0.5 seconds; it is however 9.7505 times faster than the 6.1048 seconds.

As a recommendation for future work, the generated hardware model of the AGPC algorithm should be combined with other shared memory registers blocks as well as other blocks from the Simulink and System Generator block libraries to build and test the complete AGPC algorithmic model with memory interfaces. Additional work should also be directed towards the generation of a AGPC algorithmic model as a co-processor intellectual property (IP) core (*pcore*) from the System Generator model of the Synthesized AGPC algorithmic model and the integration of this *pcore* as a co-processing hardware with a predesigned embedded soft processor core (MicroBlaze) or hard processor core (PowerPC™440) system.

References

- [1] R. S. Russel, “Non-linear F-16 Simulation using Simulink and Matlab, Ver. 1.0”, Technical Report, University of Minnesota, United States of America, 2003.
- [2] B. L. Stevens and F. L. Lewis, “Aircraft Control and Simulation”. 2nd ed., New York: John Wiley & Sons, Inc., 2003.
- [3] V. A. Akpan and G. D. Hassapis, “Training dynamic feedforward neural networks for online nonlinear model identification and control applications”. *International Reviews of Automatic Control: Theory & Applications*, vol. 4, no. 3, pp. 335-350, 2011.

- [4] V. A. Akpan and G. D. Hassapis, "Nonlinear model identification and adaptive model predictive control using neural networks", *ISA Transactions*; vol. 5, no. 2, pp. 177–94, 2011.
- [5] V. A. Akpan, "Development of new model adaptive predictive control algorithms and their implementation on real-time embedded systems", Aristotle university of Thessaloniki, GR-54124, Thessaloniki, Greece, Ph.D. Dissertation, 517 pages, July, 2011. Available [Online]: <http://invenio.lib.auth.gr/record/127274/files/GRI-2011-7292.pdf>.
- [6] L. T. Nguyen, M. E. Ogburn, W. P. Gilbert, K. S. Kibler, P. W. Brown and P. L. Deal, "Simulator Study of Stall/Post-Stall Characteristics of a Fighter Airplane With Relaxed Longitudinal Static Stability". NASA Technical Paper 1538, Dec., 1979, 233 pages.
- [7] A. Savran, R. Tasaltin and Y. Becerikli. "Intelligent adaptive nonlinear flight control for a high performance aircraft with neural networks". *ISA Transactions*, vol. 45, no. 2, pp. 225-247, 2006.
- [8] M. Hanhila, T. Mantere and J. T. Alander, "FPGA–implementation of PID-controller by differential evolution optimization", *DE GRUYTER: Open Engineering*, vol. 8, pp. 395-402, 2017.
- [9] J. M. Maciejowski, "Predictive Control with Constraints". England: Pearson Education Limited, 2002.
- [10] E. F. Camacho and C. Bordons, "Model Predictive Control". 2nd ed., London: Springer-Verlag, 2007.
- [11] J. E. Normey-Rico and E. F. Camacho, "Control of Dead-Time Processes". London: Springer-Verlag, 2007.
- [12] K. V. Ling, B. F. Wu and J. M. Maciejowski, "Embedded model predictive control (MPC) using a FPGA". In *Proceedings of the 17th World Congress, The International Federation of Automatic Control (IFAC)*, Seoul, Korea, July 6-11, 2008, pp. 15250-15255, 2008.
- [13] E. Hartley, "Model predictive control on an FPGA: Aerospace and Space Scenarios", Workshop on Embedded Optimization EMOPT 2014, IMT Lucca, University of Cambridge, United Kingdom, 8th September, 2014, pp. 1-53. Available [Online]: <https://www.imtlucca.it/emopt-14/Slides/hartley.pdf>.
- [14] Y. Kondratenko and E. Gordienko, "Implementation of the neural networks for adaptive control system on FPGA" *Annals of DAAAM for 2012 & Proceedings of the 23rd International DAAAM Symposium*, Vienna-Austria, vol. 23, no. 1, pp. 1-4, 2012.
- [15] S. Gerksić and B. Pregelj, "Finite-word-length FPGA implementation of model predictive control for ITER resistive wall mode control", *Fusion Engineering and Design*, vol. 112480, 2021. DOI: <https://doi.org/10.1016/j.fusengdes.2021.112480>.
- [16] K. Mohamed, A. E. Mahdy and M. Refai, "Model predictive control using FPGA", *International Journal of Control Theory and Computer Modeling*, vol. 5, no. 2, pp. 1-14, 2015.
- [17] S. Lucia, D. Navarro, O. Lucia, P. Zometa and R. Findeisen, "Optimized FPGA implementation of model predictive control for embedded systems using high level synthesis tool", *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 137-145, 2018.
- [18] Y. Shoukry, M. W. El-Kharashi and S. Hammad, "MPC-on-chip: An embedded GPC coprocessor for automotive active suspension systems". *IEEE Embedded Systems Letters*, vol. 2, no. 2, pp. 31-34, 2010.
- [19] P. Vouzis, L. G. Bleris, M. Arnold and M. V. Kothare, "A custom-made algorithmic-specific processor for model predictive control". In *Proceedings of the International Symposium on Industrial Electronics*, Montreal, Canada, 9-13 June, 2006.
- [20] E. N. Hartley, J. Jerez, A. Suardi, J. Maciejowski, E. C. Kerrigan and G. A. Constantinides, "Predictive control using an FPGA with application to aircraft control", *IEEE Transactions on Control Systems Technology*, vol. 22, no. 3, pp. 1006-1017, 2014. DOI: 10.1109/TCST.2013.2271791.
- [21] R. DeMott, "Development of a flexible FPGA-based platform for flight control system research", Ph.D. Theses and Dissertation, Graduate School, Virginia Commonwealth University, Richmond, Virginia, United States of America, pp. 1-157, 2010. Available [Online]: <https://core.ac.uk/download/pdf/51289440.pdf>.
- [22] M. Aboelaze, O. El-Debb, A. E. Mansour and M. A. Ghazy, "FPGA implementation of a satellite altitude control using variable structure control", In *Proceedings of the 2nd International Conference on Computer Science and Engineering (MIC-Computing 2014)*, Milan Italy, pp. 1-6, 2014. DOI: 10.13140/RG.2.1.4462.5448.
- [23] H. U. Azad, D. V. Lazic and W. Shahid, "FPGA based longitudinal and lateral controller implementation for a small UAV", *World Academy of Science, Engineering and Technology*, vol. 46, pp. 10-22, 2010.
- [24] L. Kalra and C. Georgakis, "Effects of process nonlinearity on the performance of linear model predictive controllers for the environmentally safe operation of a fluid catalytic cracking unit". *Industrial Engineering Chemical Research*, vol. 33, pp. 3063-3069, 1994.
- [25] B. H. Fletcher, "FPGA embedded processors: Revealing true system performance". *Embedded Training Program, Embedded System Conference*, San Francisco, USA, 6-10 Mar., 2005, ETP–367, pp. 1-18, 2005.
- [26] V. A. Akpan, "Model-based embedded-processor systems design methodologies: Modeling, syntheses, implementation and validation", *African Journal of Computing and ICTs*, vol. 5, no. 1, pp. 1-26, 2012.
- [27] V. A. Akpan, "Hard and soft embedded FPGA processor systems design: Design considerations and performance comparisons", *International Journal of Engineering and Technology*, vol. 3, no. 11, pp. 1000-1020, 2013.
- [28] V. A. Akpan, "An FPGA realization of integrated embedded multi-processors system: A hardware-software co-design approach", *Journal of Advanced Research in Embedded System*, vol. 2, no. 1, pp. 1-27, 2015.
- [29] V. A. Akpan, "FPGA-in-the-Loop implementation of an adaptive matrix inversion algorithmic co-processor: A dual-processor system design", *Journal of Advanced Research in Embedded System*, vol. 2, no. 1, pp. 28-63, 2015.
- [30] H. K. Khalil, "Nonlinear Systems". Upper Saddle River, NJ: Prentice-Hall, 1996.

- [31] M. Morari and E. Zafiriou, “Robust Process Control”, Englewood Cliffs, NJ: Prentice-Hall, 1989.
- [32] M. Nørgaard, O. Ravn, N. K. Poulsen and L. K. Hansen, “Neural Networks for Modelling and Control of Dynamic Systems: A Practitioner’s Handbook”, London: Springer-Verlag, 2000.
- [33] D. F. Anderson and S. Eberhardt, “Understanding Flight”, New York, U.S.A.: McGraw-Hill, 2001.
- [34] D. F. Anderson and S. Eberhardt, “Understanding Flight”, 2nd Ed., New York, U.S.A.: McGraw-Hill, 2010.
- [35] G. J. J. Ducard, “Fault-Tolerant Flight Control and Guidance Systems: Practical Methods for Small Unmanned Aerial Vehicles”, London: Springer-Verlag Limited, 2009.
- [36] R. C. Nelson, “Flight Stability and Automatic Control”, New York: McGraw-Hill, Inc, 1989.
- [37] The MathWorks Inc., MATLAB® & Simulink® R2021a, Natick, USA. www.mathworks.com.
- [38] Xilinx Inc., “Xilinx Software Design Suite 14.5”, (2022). USA. www.xilinx.com.
- [39] P. Meloni, S. Secchi and L. Raffo, “An FPGA-based framework for technology-aware prototyping of multicore embedded architectures”. IEEE Embedded Syst. Letters, vol. 2, no. 1, pp. 5-9, 2010.
- [40] Xilinx Inc., “Xilinx AccelDSP Synthesis Tool”, User Guide, v11.4, December 2, 2009, pp. 1-222.
- [41] Xilinx Inc., “Xilinx System Generator for DSP”, User Guide, v12.1, April 19, 2014, pp. 1-414.
- [42] Sjöberg, J and Ljung, L. (1995). “Overtraining, regularization, and searching for minimum in neural networks”. International Journal of Control, vol. 62, pp. 1391-1408.

Biography



Vincent Andrew Akpan obtained his Ph. D. degree in Electrical & Computer Engineering from the Aristotle University of Thessaloniki (AUTH), Thessaloniki, Greece in 2011.

He is currently an Associate Professor and Acting Head of Biomedical Technology Department, FUTA, Akure, Nigeria. His research interest is in electronic instrumentation, intelligent control, embedded systems as well as mechatronics & rehabilitation engineering. He is the co-author of a book and has authored and/or co-authored more than 60 articles in refereed journals and conference proceedings.

Dr. Akpan is a member of several professional societies. He is also a Fellow of CBET Nigeria and IPMD Nigeria.



Dimitrios Chasapis received his Ph.D. from Universitat Politècnica de Catalunya, Barcelona, Spain in 2019.

He is currently part of the Software research and development vehicles for New Architectures (SONAR) group of Barcelona Supercomputing Center (BSC), holding the position of recognized researcher. His main academic work focuses on high performance computer architectures and runtime systems, performance analysis tools, parallel programming languages and machine learning. He has authored and/or co-authored several articles in refereed journal and conference proceedings.

He is a member of several professional societies.



George Dimitriou Hassapis received his Ph.D. degrees from the Control Systems Centre, University of Manchester, U.K.

Since January 2019, he has been an Emeritus Professor of Electrical and Computer Engineering, AUTH, Greece. His research interests are focused on Computer Systems Architecture, Control & Intelligent Systems, Real-Time Embedded Systems, e-Learning Educational Technology, and Biomedical Engineering. He is the author of two books and he has authored and/or co-authored more than 120 articles in refereed journals, conference proceedings and book chapters.

He is a SMIEEE, CENG, Emeritus Member, Technical Chamber of Greece, and many other professional societies.