Analog Integr Circ Sig Process
DOI 10.1007/s10470-014-0263-7

FPGA implementation of high-speed neural network for power amplifier behavioral modeling

Mohammed Bahoura

Received: 18 July 2013 / Revised: 19 January 2014 / Accepted: 25 January 2014

Abstract In this paper, a high-speed pipelined architecture of dynamic neural network is proposed for power amplifier behavioral modeling. This architecture is implemented on field programmable gate array (FPGA) using Xilinx system generator and Virtex-6 FPGA ML605 Evaluation Kit. The novelty of the proposed architecture is that it provides higher operating frequency, lower output latency, and less required resources. These improvements are obtained by reducing the bit-width data and by efficiently redistributing the inserted pipelining delays. The new pipelined architecture is evaluated and compared to the conventional and pseudo-conventional ones in terms of the resource utilization, the maximum operating frequency, and the modeling performances using the 16-QAM modeled test signal. This architecture is verified using JTAG hardware co-simulation both for single step and free-running clock modes.

Keywords Adaptive modeling · Neural network · Power amplifier · Memory effects · Pipelining · FPGA

1 Introduction

During the last decades, artificial neural networks (ANNs) have been widely used in different research fields. They are most commonly used for prediction, pattern recognition, process control, etc. Different kinds of ANNs are proposed in the literature but the multi-layer perceptron (MLP) is probably the most popular and the simplest neural network. The back-propagation (BP) algorithm is the most widely used technique for supervised learning of this kind of ANNs. However, the software implementation of an ANN-based system and its learning algorithm is usually very time-consuming because it consists of massively parallel nonlinear computations. In many applications, like radio frequency (RF) power amplifier modeling and linearization [1–8], the software implementation is useful to evaluate the capabilities of the ANN-based solutions but the hardware implementation, which can takes advantage of the inherent parallelism of the ANN, remains the ultimate objective of this kind of applications.

Literature review shows that ANNs have been traditionally implemented on specific-purpose analog and digital hardware [9, 10], and recently they are implemented on field programmable gate arrays (FPGA) [11]. Although their limited gate density, the FPGAs are preferred to the specific purpose circuits because they are low-cost and reconfigurable, offering software like flexibility. To implement large size ANNs, time and neuron multiplexing are generally adopted to overcome the logic density limitation of FPGAs [11, 12]. These approaches can not fully exploit concurrency and rapidity allowed by FPGAs. On the other hand, several techniques are also proposed to approximate the non-linear activation function: CORDIC algorithm, polynomial approximation, lookup table (LUT) method, etc. [13]. In practice, there are three relevant design metrics in the hardware implementation of ANN: accuracy, resource, and processing speed [13]. However, in some applications such wireless communication systems, the processing speed is certainly more privileged than the required resources.

The RF power amplifiers (PAs), which are key components in wireless transceivers, present nonlinearity and memory effects. The nonlinearity causes amplitude and

M. Bahoura (✉)
Department of Engineering, Université du Québec à Rimouski, 300, allée des Ursulines, Rimouski, QC G5L 3A1, Canada
e-mail: Mohammed_Bahoura@uqar.ca

Published online: 12 February 2014

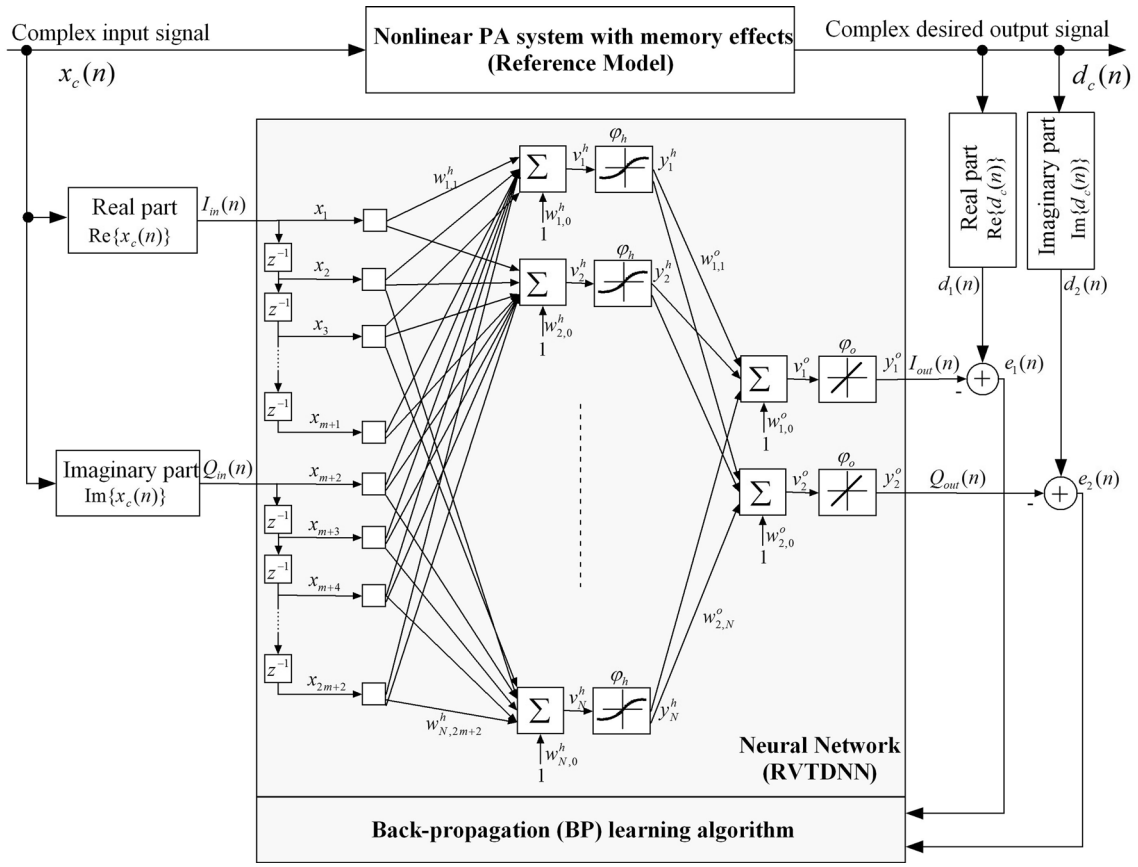

Fig. 1 Diagram of dynamic two-layer RVTDNN for PA behavioral modeling [7]

phase distortions at output of PAs and spectral regrowth that interferes with adjacent channels. Among all linearization techniques, digital predistortion (DPD) has been widely studied in the literature the last years because of its efficiently and flexibility. It is also made popular by the technological progress in digital signal processing hardware. Digital predistorter acts on the input signal by using a nonlinear system, which is an estimate of the inverse model of the PA to be linearized, to obtain an overall linear system. Several models have been proposed such as lookup table [14–16], polynomial model [16–19], Volterra series [20–22], neural network [1–8], and neural-fuzzy model [23].

The hardware implementation of DPD can be done by using application-specific integrated circuits (ASICs), digital signals processors (DSPs) or FPGAs. However, FPGAs are generally preferred to ASICs and DSPs because they are low-cost and reconfigurable, offering software like flexibility. FPGA-based solutions are already proposed for lookup table [14–16], polynomial model [16, 19, 24], Volterra series [22], and neural network [7, 8].

In this paper, we propose a high-speed fully-pipelined architecture of a parallel neural network for adaptive power amplifier modeling. The high-speed performance is obtained by the reduction of the fixed-point data format and the efficient redistribution of the pipelining delays.

2 RVTDNN-based PA models

The PA nonlinearity and memory effects are modeled using a real-valued time-delay neural network (RVTDNN), which is based on a MLP by adding tapped delay lines (TDLs) at its inputs [1]. Figure 1 presents a dynamic modeling diagram, where the RVTDNN neural network model is continuously updated to prevent any eventual change in the PA characteristics. Let us consider a neural network having N_0 inputs, one hidden layer of N_1 neurons, and N_2 outputs. At any moment n, the input data vector $\mathbf{x}$, of $N_0 = 2m + 2$ components, is constructed from the in-phase (I_{in}) and quadrature (Q_{in}) signals, and their m past values [7]:

$$\mathbf{x} = \begin{matrix}[I_{in}(n), I_{in}(n-1), \ldots, I_{in}(n-m), \\ Q_{in}(n), Q_{in}(n-1), \ldots, Q_{in}(n-m)]\end{matrix} \quad (1)$$

where m is also called the memory depth ($m = 2$).

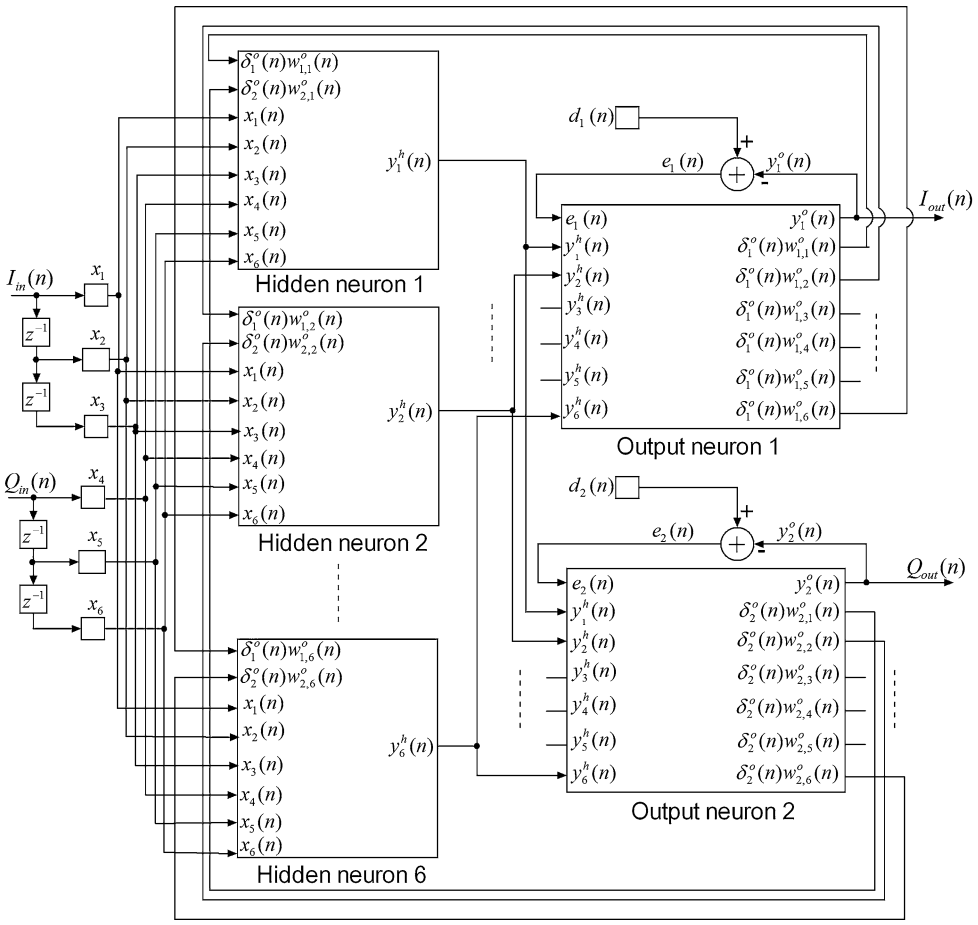

Fig. 2 Conventional architecture for a RVTDNN6 PA model that is characterized by $N_0 = 6$ inputs, $N_1 = 6$ hidden neurons, and $N_2 = 2$ output neurons. Details on neurons and their corresponding synoptic weights and biases update are given in Fig. 3

2.1 Conventional architecture

This architecture implements the mathematical equations describing the original RVTDNN model [1] and its BP learning algorithm [25]. Figure 2 presents the conventional architecture (RVTDNN6-con) of a neural network that is characterized by 6 inputs, 6 hidden neurons, and 2 output neurons. Details on neuron blocks are given in Fig. 3 that also shows how the synoptic weights and biases are calculated and updated.

2.1.1 Forward inputs propagation

As shown in Fig. 1, the output $y_j^h(n)$ of a given hidden neurone j is obtained by performing a weighted sum $v_j^h(n)$ that is transferred by a nonlinear function φ_h [7].

$$y_j^h(n) = \varphi_h\left(v_j^h(n)\right) \tag{2}$$

and

$$v_j^h(n) = \sum_{i=1}^{N_0} w_{j,i}^h(n)x_i(n) + w_{j,0}^h(n) \qquad j = 1,\ldots,N_1 \tag{3}$$

where $w_{j,i}^h(n)$ is the synaptic weight connecting the ith node from the input layer to the jth neuron of the hidden layer, and $w_{j,0}^h(n)$ is the bias of the jth hidden neuron. The sum of the weighted inputs is implemented using a binary tree of two-input adders to efficiently minimize the critical path [26] (Fig. 3).

In the same manner, the output of each neuron k, in the output layer, is given by:

$$y_k^o(n) = \varphi_o\left(v_k^o(n)\right) \tag{4}$$

and

$$v_k^o(n) = \sum_{j=1}^{N_1} w_{k,j}^o(n)y_j^h(n) + w_{k,0}^o(n) \qquad k = 1,\ldots,N_2 \tag{5}$$

where φ_o is the activation function, $w_{k,j}^o(n)$ is the synaptic weight connecting the output of the jth neuron in the hidden layer to the kth neuron of the output layer, and $w_{k,0}^o(n)$ is the bias of the kth neuron. In this paper, we use an *hyperbolic tangent* function $\varphi_h(x) = (1 - e^{-2x})/(1 + e^{-2x})$ for the hidden layer and a *linear* function $\varphi_o(x) = x$ for the output layer, which is a common choice for power

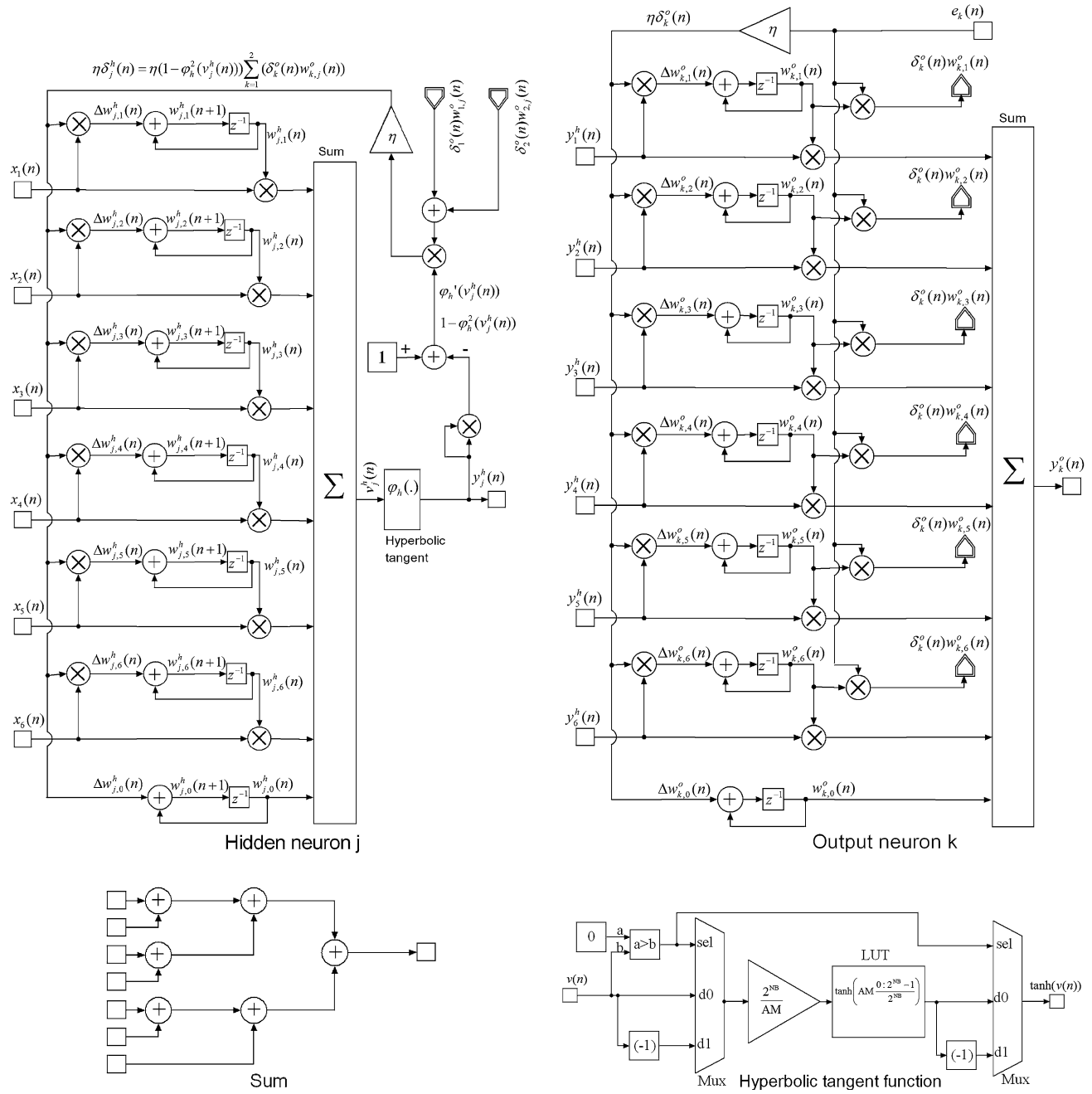

Fig. 3 The hidden and output neurons of the conventional RVTDNN are given on *top-left* and *top-right*, respectively. The sum and the hyperbolic tangent blocks are described in *bottom*

amplifier modeling [1, 4, 5]. As shown in Fig. 3, $\varphi_h(x)$ is implemented using a look-up table (LUT) schema that takes advantage from the asymmetric characteristic of this function. For the jth hidden neuron, positive value of $v_j^h(n)$ is multiplied by $2^{\mathrm{NB}}/\mathrm{AM}$ to convert it to an address value of unsigned NB-bit fixed-point format, where AM is the maximum amplitude of $v_j^h(n)$. The output corresponds to the word addressed by the given input. The activation function output of negative value is obtained using asymmetry $(\tanh(-v) = -\tanh(v))$. However, $\varphi_o(x)$ is implemented by a simple wire because it is linear.

2.1.2 Backward error propagation

At the nth iteration (i.e. presentation of the nth training example), the error signal at the output of the kth neuron of the output layer is defined by

$$e_k(n) = d_k(n) - y_k^o(n) \tag{6}$$

where $d_k(n)$ and $y_k^o(n)$ are the desired and the actual outputs of this neuron, respectively. The total error energy $E(n)$ [7] at the output layer can be reduced by updating the connection weights using the gradient descent method [25].

$$\Delta w_{k,j}(n) = -\eta \frac{\partial E(n)}{\partial w_{k,j}(n)} \tag{7}$$

where $(0 < \eta < 1)$ is the *learning rate*. A small value of η can guarantee convergence but involves a slow learning. On the other hand, a large value of this parameter involves rapid learning but can lead to oscillation or even divergence. The next value of $w_{k,j}(n)$ is then given by:

$$w_{k,j}(n+1) = w_{k,j}(n) + \Delta w_{k,j}(n) \tag{8}$$

2.1.2.1 Output layer The connection weights in the output layer are adjusted by [25]:

$$\Delta w_{k,j}^o(n) = \eta \delta_k^o(n) y_j^h(n) \tag{9}$$

where $y_j^h(n) = 1$. As a *linear* function is used [7], the local gradient for the neurons of this layer $\delta_k^o(n)$ is defined by:

$$\delta_k^o(n) = e_k(n) \tag{10}$$

2.1.2.2 Hidden layer The synoptic weights in the hidden layer are adjusted using:

$$\Delta w_{j,i}^h(n) = \eta \delta_j^h(n) x_i(n) \tag{11}$$

where $x_0(n) = 1$. As an *hyperbolic tangent* function is used [7], the local gradient $\delta_j^h(n)$ in this layer is defined by:

$$\delta_j^h(n) = (1 - \varphi_h^2(v_j^h(n))) \sum_{k=1}^{N_2} \delta_k^o(n) w_{k,j}^o(n) \tag{12}$$

In the conventional architecture, the LUT-based computation of the activation function must be implemented only by the distributed memory. In fact, the distributed memory doesn't require delays while the block RAM memory involves at least one delay unit [27]. On the other hand, only some slices of the FPGA can use their LUTs as distributed memory. Consequently, conventional architecture can be implemented only for RVTDNN models requiring small size LUTs (few neurons in hidden layer, reduced size data, etc.).

2.2 Pseudo-conventional architecture

To implement RVTDNN models requiring large size LUTs (more neurons in hidden layer with more accurate computation of the activation function), the block RAM memories must be used instead of the distributed memories. This approach involves delays in the forward and backward propagation paths. When only one delay is inserted, the resulting architecture is called pseudo-conventional.

Figure 4 presents the pseudo-conventional architecture (RVTDNN6-pse). A delay unit (z^{-1}) inserted by the hyperbolic tangent function is transmitted to the neuron outputs $y_j^h(n-1)$ in hidden layer, neuron outputs $y_k^o(n-1)$ in output layer, and output errors $e_k(n-1)$. Details on hidden and output neurons are given in Fig. 5, where the hyperbolic tangent function is implemented using block RAM memories.

2.2.1 Forward inputs propagation

For this architecture, the output $y_j^h(n-1)$ of the hidden neuron j and its weighted sum $v_j^h(n-1)$ can be defined from (2) and (3), respectively, by adding one delay unit. Similarly, the output $y_j^o(n-1)$ of the output neuron k can be derived from (4) by inserting one delay, where its weighted sum is defined by:

$$v_k^o(n-1) = \sum_{j=1}^{N_1} w_{k,j}^o(n) y_j^h(n-1) + w_{k,0}^o(n) \tag{13}$$

2.2.2 Backward error propagation

The error signal $e_k(n-1)$ at the output neuron k is obtained by synchronizing desired an actual outputs. It can by defined from (6) by inserting one delay unit.

2.2.2.1 Output layer By synchronizing signals at the output layer, the connection weights and biases are updated using:

$$\Delta w_{k,j}^o(n) = \eta \delta_k^o(n-1) y_j^h(n-1) \tag{14}$$

where the delayed local gradient $\delta_k^o(n-1)$ can be derived from (10) by inserting one delay unit.

2.2.2.2 Hidden layer By synchronizing signals at the hidden layer, the connection weights and biases are updated using:

$$\Delta w_{j,i}^h(n) = \eta \delta_j^h(n-1) x_i(n-1) \tag{15}$$

where the delayed local gradient $\delta_j^h(n-1)$ is defined by

$$\delta_j^h(n-1) = (1 - \varphi_h^2(v_j^h(n-1))) \sum_{k=1}^{N_2} \delta_k^o(n-1) w_{k,j}^o(n) \tag{16}$$

The pseudo-conventional architecture allows us to implement larger size RVTDNN PA models because the size of block RAM memories (14,976 Kb) on the used FPGA is larger than that of the distributed memories (3,650 Kb). For small size neural networks, the pseudo-conventional architecture can be implemented either by block RAM or

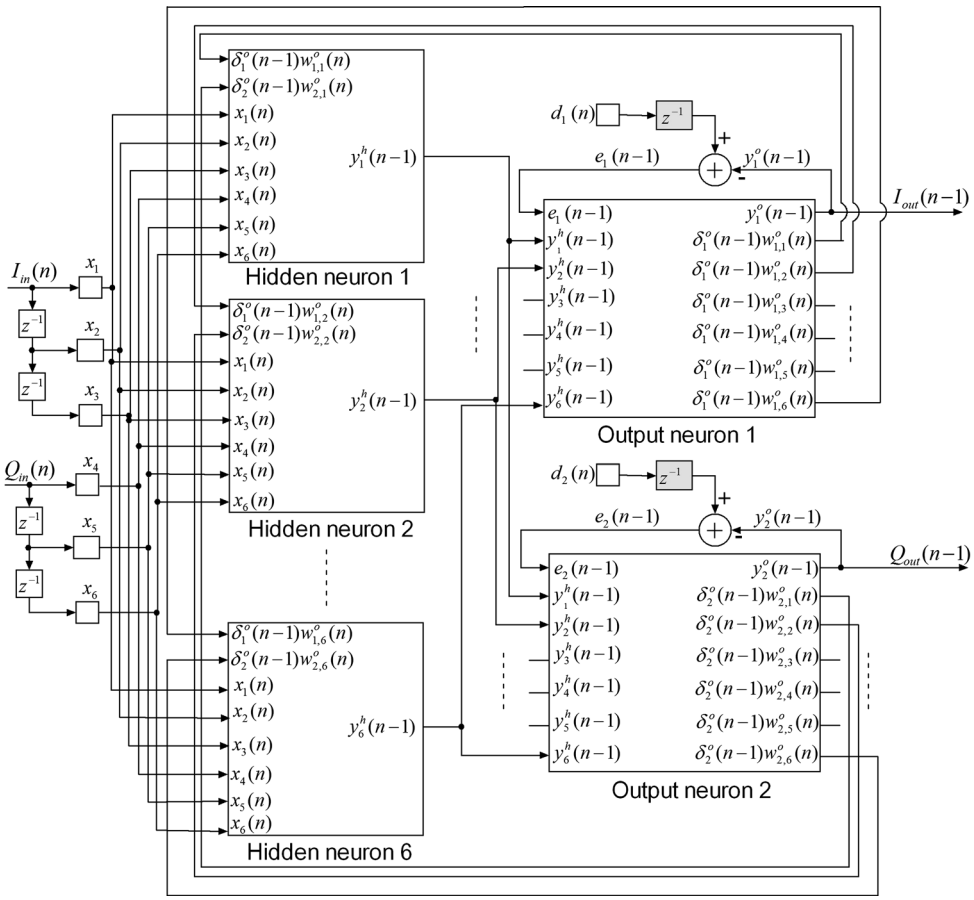

Fig. 4 Pseudo-conventional architecture for a RVTDNN6 PA model, where the *dashed boxes* represent the inserted delays. Details on neurons and their corresponding synoptic weights and biases update are given in Fig. 5

distributed memories. Note that architectures based on block RAM memories are faster than those based on distributed memories [27]. On the other hand, computation speed can be greatly improved in the pipelined architecture that is detailed in the next subsection. The pipelining process consists to reduce the critical path by inserting delays between the computational elements (multipliers and adders) of the implemented system. However, this process presents two main drawbacks: increasing the number of the used latches and the output latency of the given system [28].

2.3 Pipelined architecture

The pipelining approach, based on the delayed BP learning algorithm [27], has been successfully applied to RVTDNN PA model with 36-bit fixed-point format (FIX36WT) leading to a pipelined architecture 6.5 faster than the pseudo-conventional one [8]. However, this architecture involves an output latency of 24 samples, a delay of 32 samples to update the synoptic weights in the output layer, and a delay of 48 samples to update those of the hidden layer [8]. In the present paper, a new architecture is proposed to improve the performance of the pipelined model. The new pipelined architecture presents higher operation frequency and lower output latency. These performance are obtained by reducing the data bit-width to 24 (FIX24RS) and efficiently redistributing the inserted delays. The proposed architecture presents an output latency of 16 samples, a delay of 20 samples to update the synoptic weights in the output layer, and a delay of 28 samples to update those of the hidden layer. As for the delayed least mean square (DLMS) algorithm [26, 29], it is shown that the delay in the weight update rules has minor effect on the stead-state behavioral if the learning rate (η) is within certain bounds [27]. For a large delay in the synoptic weights and biases update, the value of η in the pipelined architecture must be reduced (compared to that in the conventional one) to guarantee stability.

Figure 6 presents a pipelined architecture (RVTDNN6-pip). Inserted delays (dashed boxes) between components affect outputs of the hidden layer $y_j^h(n-9)$, those of the output layer $y_k^o(n-16)$, and the output errors $e_k(n-16)$. The output latency improvement (16 instead of 24) is attributed to the reduction of number of delays needed to fully-pipeline the multiplier blocks. In fact, the pipelining of one multiplier block requires seven delays for the 36-bit fixed-point data [8] and only 4 delays for the 24-bit one [7]. Figure 7 gives more details on how neurons are implemented. Synoptic weights

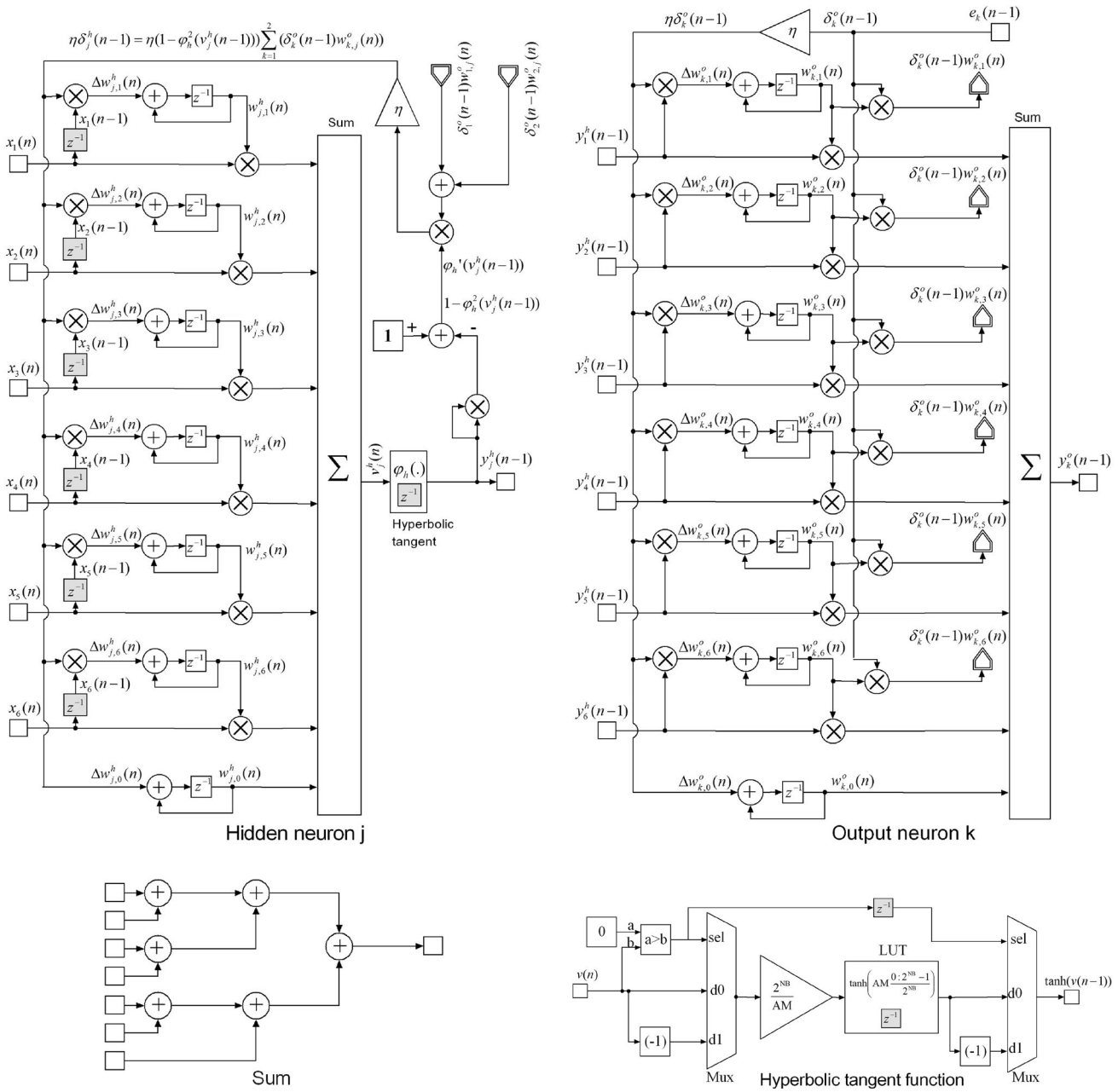

Fig. 5 The hidden and output neurons of the pseudo-conventional RVTDNN are given on *top-left* and *top-right*, respectively, where the *dashed boxes* represent the inserted delays. The sum and the hyperbolic tangent blocks are described in *bottom*

and biases are updated by synchronizing inserted delays in different forward and backward propagation paths.

2.3.1 Forward inputs propagation

For this architecture, the output $y_j^h(n-9)$ of the hidden neuron j and its weighted sum $v_j^h(n-9)$ can be defined from (2) and (3), respectively, by adding 9 delay units. Similarly, the output $y_j^o(n-16)$ of the output neuron k can be derived from (4) by inserting 16 delay units, where its weighted sum is defined by:

$$v_k^o(n-16) = \sum_{j=1}^{N_1} w_{k,j}^o(n-7) y_j^h(n-16) + w_{k,0}^o(n-7) \tag{17}$$

2.3.2 Backward error propagation

The error signal $e_k(n-16)$ at the output neuron k is obtained by synchronizing desired an actual outputs. It can by defined from (6) by inserting 16 delay units.

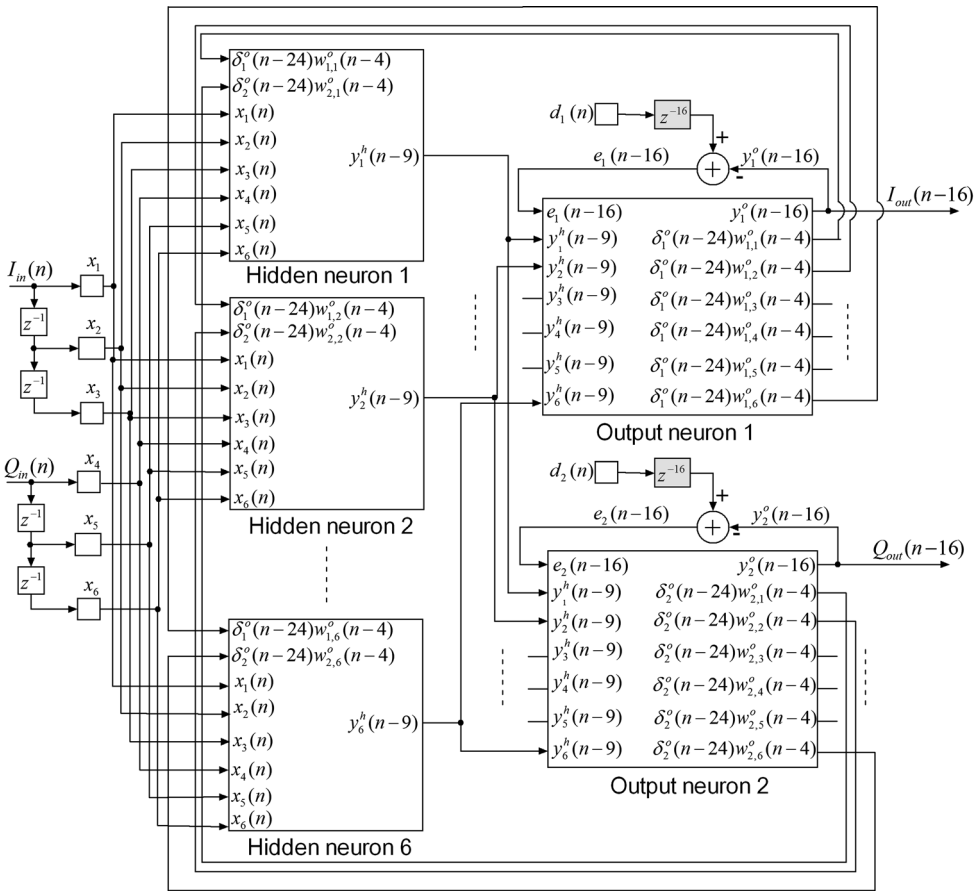

Fig. 6 Pipelined architecture for a RVTDNN6 PA model, where the *dashed boxes* represent the inserted delays. Details on neurons and their corresponding synoptic weights and biases update are given in Fig. 7

2.3.2.1 Output layer By inserting an appropriate number of delays, the connection weights in the output layer are updated using :

$$\Delta w^o_{k,j}(n) = \eta \delta^o_k(n-20) y^h_j(n-20) \tag{18}$$

where the delayed local gradient $\delta^o_k(n-20)$ can be derived from (10) by inserting 20 delay units.

2.3.2.2 Hidden layer By inserting an appropriate number of delays, the connection weights in the hidden layer are updated using:

$$\Delta w^h_{j,i}(n) = \eta \delta^h_j(n-28) x_i(n-28) \tag{19}$$

where the delayed local gradient $\delta^h_j(n-28)$ is defined by:

$$\delta^h_j(n-28) = (1-\varphi^2_h(v^h_j(n-28))) \sum_{k=1}^{N_2} \delta^o_k(n-28) w^o_{k,j}(n-12) \tag{20}$$

As it can be seen later in Sect. 5, the convergence speed of the pipelined architectures is not significantly affected by the inserted delays in the weights and biases update.

3 Reference model

A reference PA system based on the Wiener model is used to illustrate the modeling performances of the proposed architectures [7]. It is based on a 2nd order finite impulse response (FIR) filter defined by:

$$H(z) = 1 + 0.5z^{-2} \tag{21}$$

to model the memory effects, followed by the most popular memoryless nonlinear PA model proposed by Saleh [30]. This model is defined by the following AM/AM and AM/PM characteristics:

$$A(r(t)) = \frac{\alpha_A r(t)}{1+\beta_A r^2(t)} \tag{22}$$

$$\Phi(r(t)) = \frac{\alpha_\Phi r^2(t)}{1+\beta_\Phi r^2(t)} \tag{23}$$

where $r(t)$ stands for the envelope of the applied input signal. Typical parameter values are: $\alpha_A = 2.1587, \beta_A = 1.1517, \alpha_\Phi = 4.0033$ and $\beta_\Phi = 9.1040$.

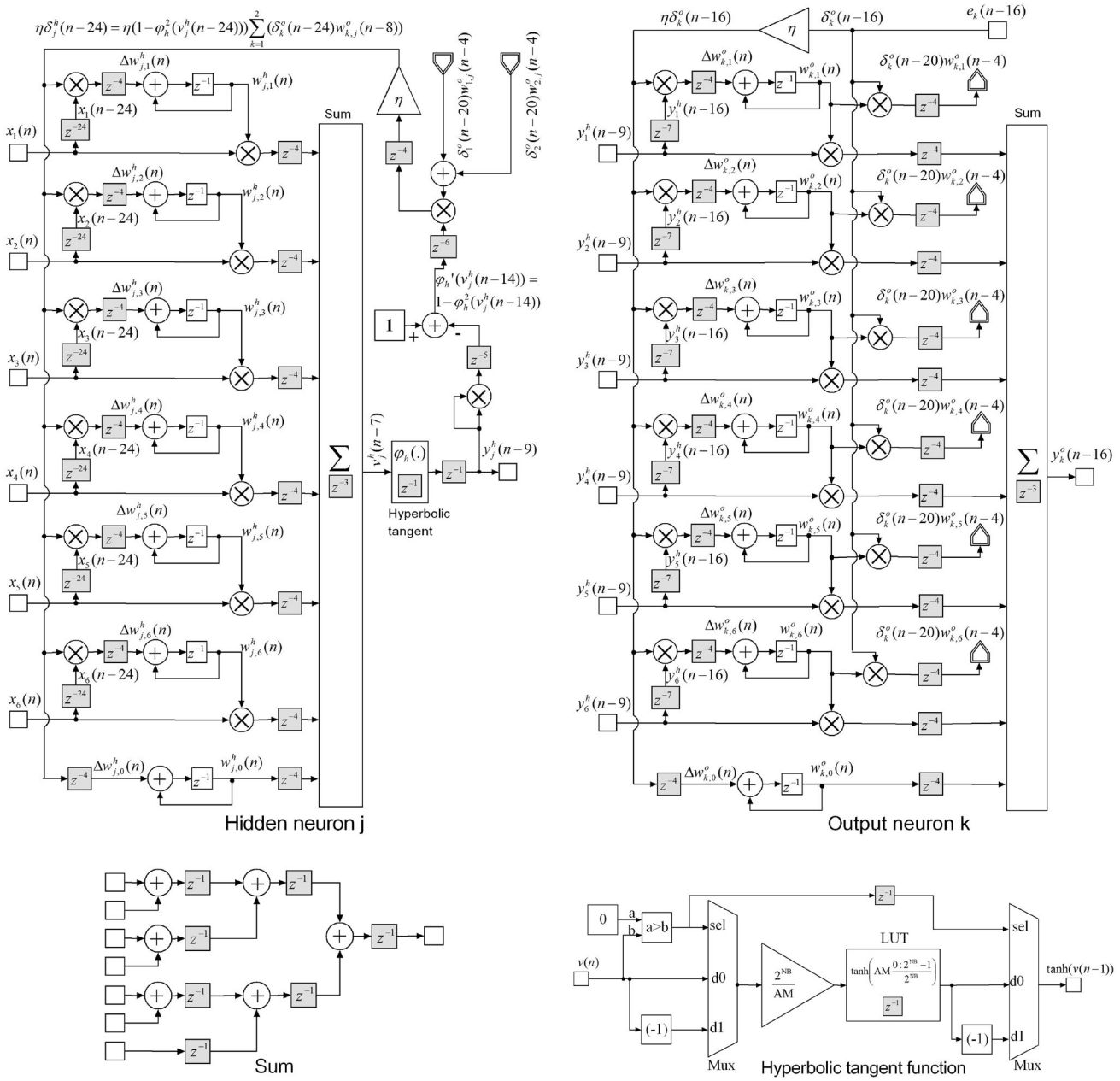

Fig. 7 The hidden and output neurons of the pipelined RVTDNN are given on *top-left* and *top-right*, respectively. The sum and the hyperbolic tangent blocks are described in *bottom*

4 FPGA implementation

Figures 8, 9 and 10 present the hardware implementation of the three above described architectures on Virtex-6 FPGA using Xilinx system generator (XSG) and ML605 Evaluation kit. Using this high-level programming tool, the three architectures are implemented by replacing arithmetic and logic blocks of Figs. 2, 3, 4, 5, 6, and 7 by their corresponding blocks from XSG blockset. For pseudo-conventional (Fig. 9) and pipelined (Fig. 10) architectures, delays are added to synchronize input and model outputs to take into account the latency at neural network output. Because of the input signal is upsampled by a factor of 8 during the raised cosine transmit filtering [7], the whole delay in each path is adjusted to 8 samples for pseudo-conventional architecture and 16 samples for the pipelined one.

For each architecture, the top-level design diagram includes SIMULINK blocks and a XSG-based system that implement the PA model. The SIMULINK blocks communicate with the XSG-based system through "*Gateway In*" and "*Gateway Out*" blocks that can be seen as an analog-to-

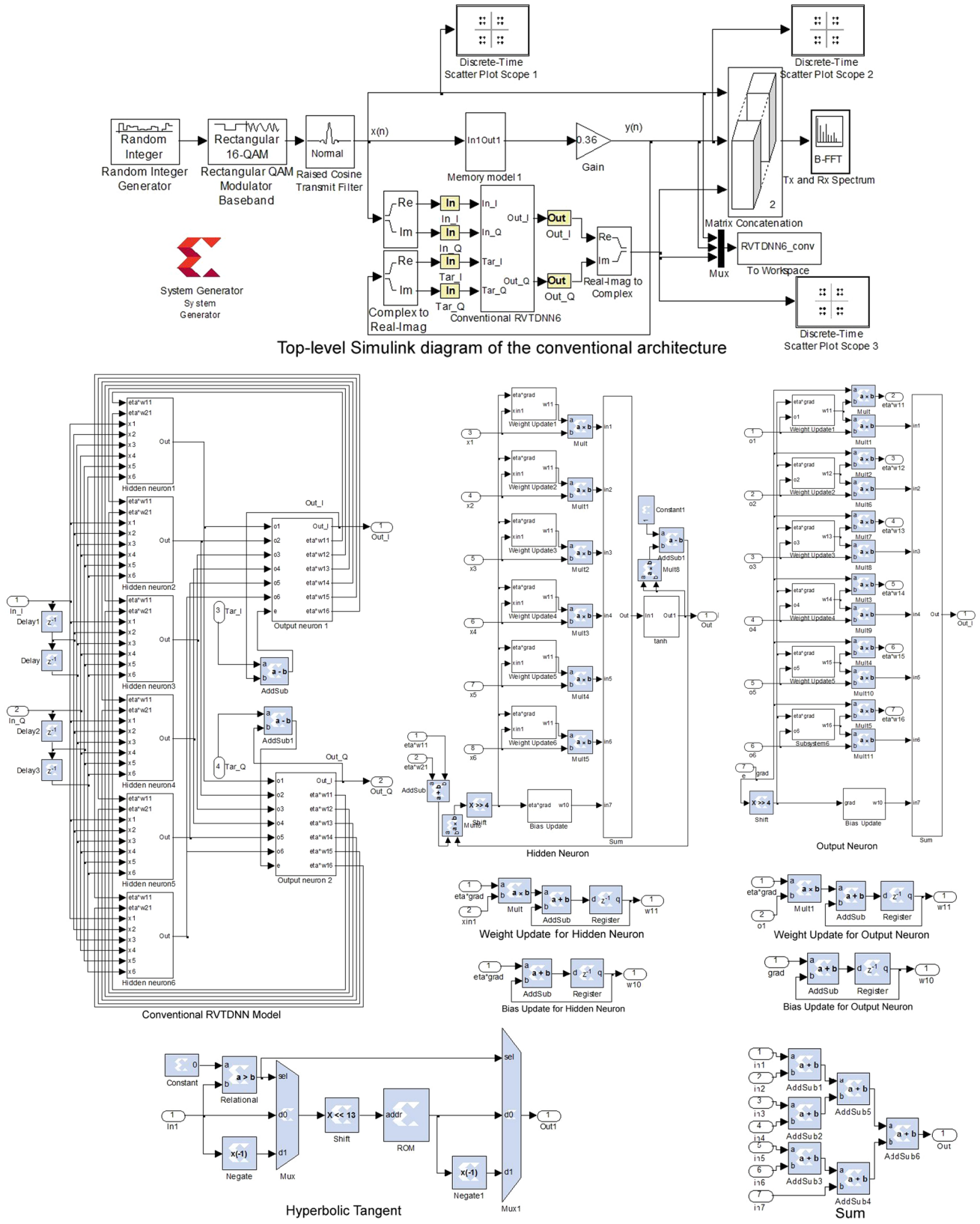

Fig. 8 Conventional architecture (RVTDNN6-con) based on Xilinx system generator (XSG) blockset for power amplifier behavioral modeling. This XSG-based architecture corresponds to the one described in Figs. 2 and 3

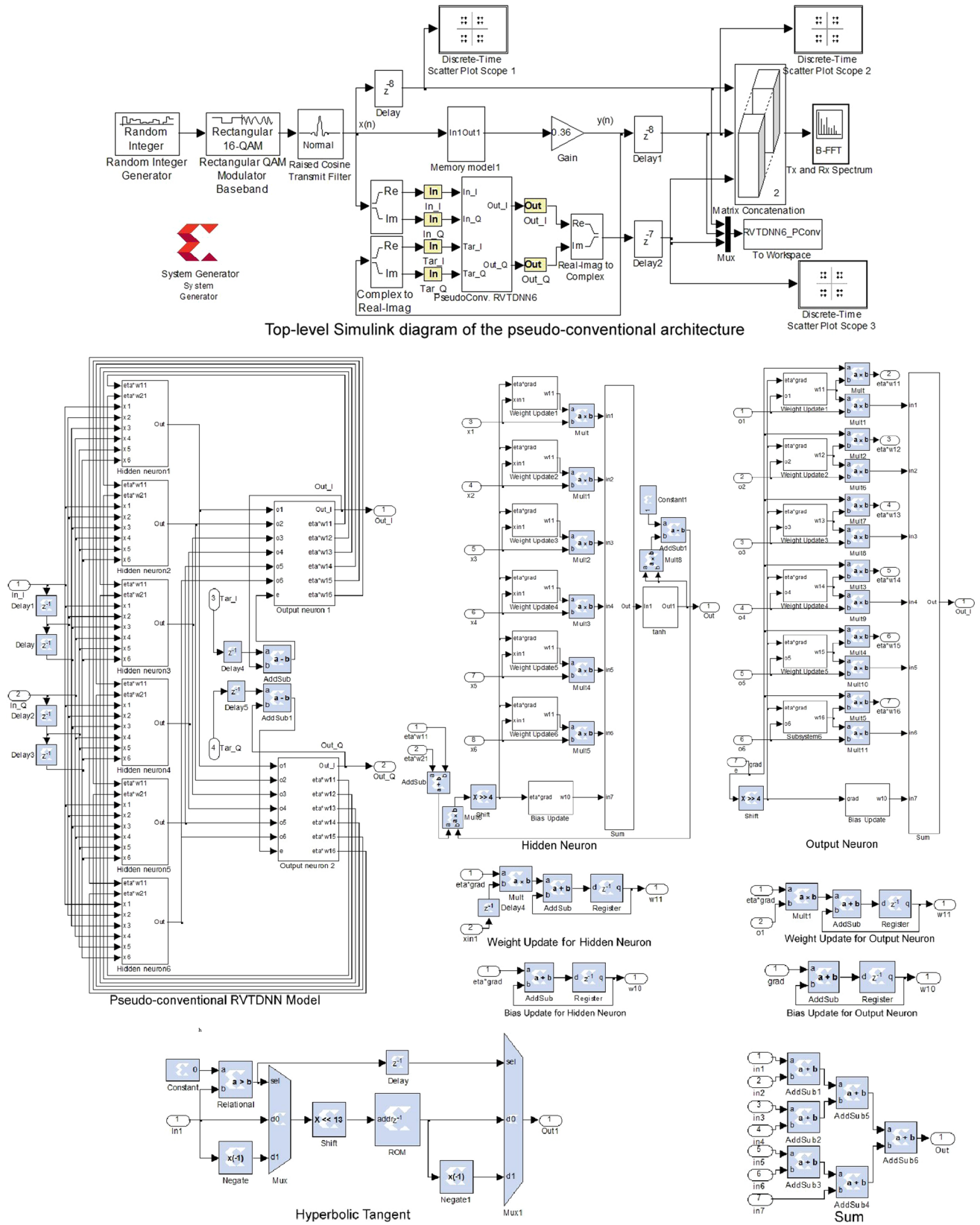

Fig. 9 Pseudo-conventional architecture (RVTDNN6-pse) based on Xilinx system generator (XSG) blockset for power amplifier behavioral modeling. This XSG-based architecture corresponds to the one described in Figs. 4 and 5

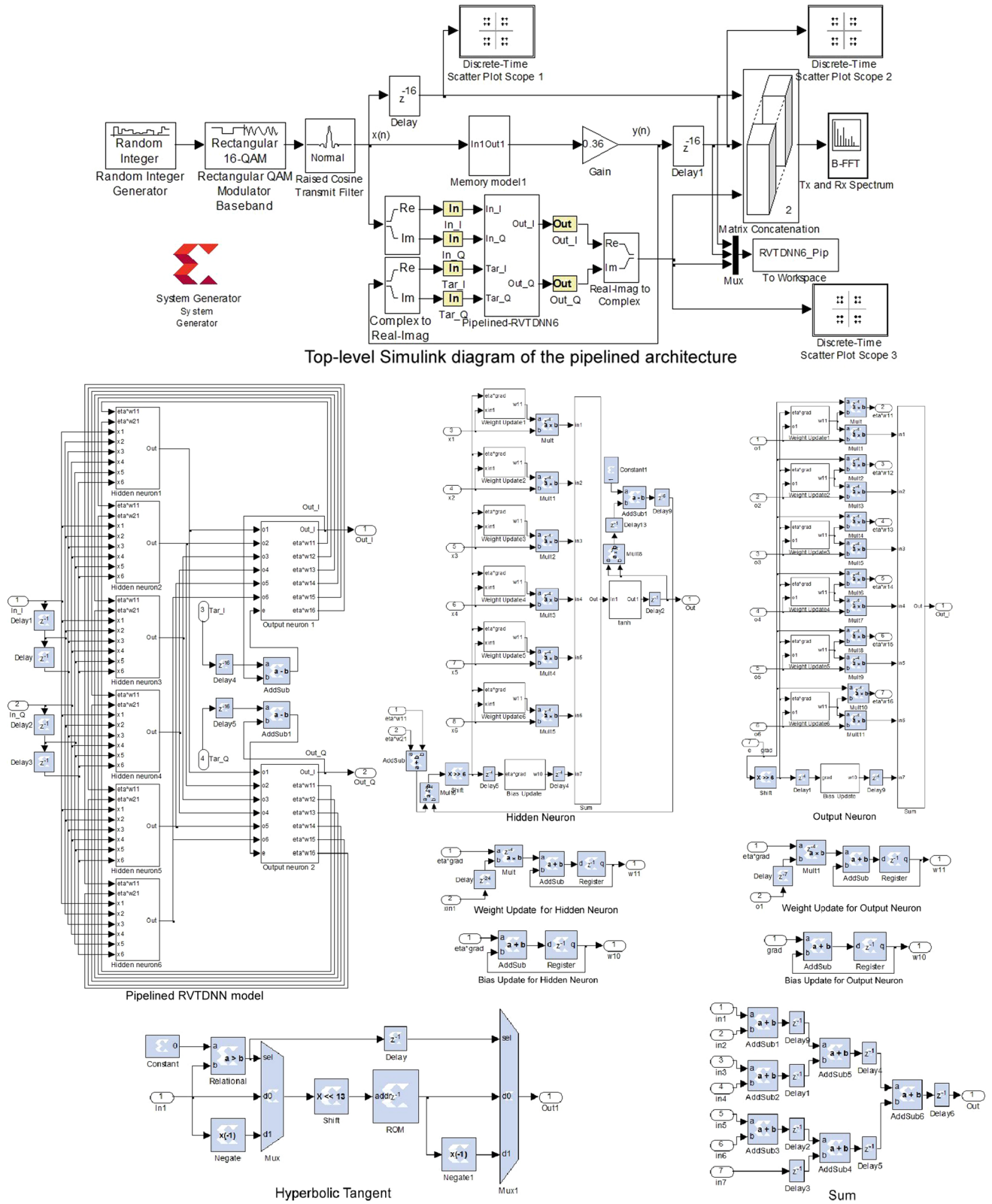

Fig. 10 Pipelined architecture (RVTDNN6-pip) based on Xilinx system generator (XSG) blockset for power amplifier behavioral modeling. This XSG-based architecture corresponds to the one described in Figs. 6 and 7

digital converter (ADC) and a digital-to-analog converter (DAC), respectively. Figs. 8, 9 and 10 present also more details about the computing subsystems used in the forward and backward propagation paths. Data in almost XSG blocks are quantized using 2′ complement signed 24-bit fixed-point format having 20 fractional bits (FIX24RS) with advanced quantization and overflow options [7]. However, only 1-bit format is used to represent the 0 and 1 constant values used to compute the hyperbolic tangent and its derivative, respectively [8]. The hyperbolic tangent function of each neuron is calculated using a Xilinx ROM block of 2^{NB} samples (NB = 15). Experimental tests show that the dynamic range of the hyperbolic function input is generally limited between −4 and 4. To convert a given positive input value ($v_j^h(n)$) to its corresponding address, a 13-bit left shifter is used rather than multiplication by $2^{NB}/AM = 2^{13}$. In conventional and pseudo-conventional architectures, the output of the 13-bit left shifter block must be saturated when its input eventually exceeds the maximum amplitude value (AM = 4) to ovoid erroneous outputs. This overflow option require additional logic components that makes longer the critical path for these architectures. However, this saturation option is not required for the pipelined architecture because of its lower learning rate η and its delayed synoptic weights update. In the conventional architecture, the activation function can be implemented only by distributed memory, but it can be implemented either by distributed and block RAM memories in the pseudo-conventional and pipelined ones. This is because the distributed memory does not require delay units, while the block RAM one involves at least one delay unit [27]. Instead of using a multiplier, a p-bit right shifter is used to implement the multiplication by the learning rate ($\eta = 2^{-p}$). For conventional and pseudo-conventional architectures, the value $\eta = 2^{-4}$ allows a good tradeoff between stability and convergence. Because of its delayed synoptic weights update, the pipelined one requires a smaller value $\eta = 2^{-6}$ to guarantee convergence and stability [27].

Table 1 gives the resource requirement and the maximum operating frequency (MOF) for conventional, pseudo-conventional and pipelined architectures of the RVTDNN6 network depending on the use of the distributed memory or the block RAM memory. There are only five combinations because the conventional architecture can not use block RAM memory that requires at least one delay unit. Note that all architectures require 240 DSP48E1s because they have the same number of multipliers. On the other hand, architectures based on the block RAM memory require 132 RAMB36E1s instead of 0 for those based on the distributed memory. However, architectures based on the distributed memory use much more logic slices (about 37,000 instead of about 5,000). It is obvious that is impossible to implement a conventional architecture of more than six hidden neurons on the used Virtex-6 FPGA (RVTDNN6-con used more than 99% of the available Slices). The pipelined architectures require more Flip-flops than the conventional one because of the inserted delays. The best MOF value of 212.865 MHz is obtained for the pipelined architecture based on the block RAM memories, which increases with the pipelining level. After distributing the inserted delays over the pipelined architecture, the MOF has been considerably improved (from 172.384 to 212.865 MHz) only by taking out the saturation overflow logic in the 13-bit left shifter block used in the computation of the hyperbolic tangent function.

Table 2 gives the resource requirement and the maximum operating frequency for the pseudo-conventional and pipelined architectures of the RVTDNN8, RVTDNN10, and RVTDNN15 networks that have 8, 10, and 15 hidden neurons, respectively. For the same number of neurons in the hidden layer, both pseudo-conventional and pipelined architectures, based on block RAM memory, use the same number DSP48E1s and RAMB36E1s. However, the pipelined architectures require more flip-flops because of the inserted delays. For these architectures, the increase of the number of hidden neurons is mainly limited by the number of the DSP48E1s and RAMB36E1s available on the used FPGA. The increase of the number of hidden neurons in the pipelined architectures involves a little decrease in the MOF (207.555 MHz instead of 212.865 MHz) due to a small increase of the critical path in the sum blocks of the output neurons. As it can be seen in Figs. 7 and 10, the sum blocks in the output neurons of the RVTDNN6-pip architecture are pipelined by inserting one delay after each adder for a total latency of three delays. By keeping the same latency, the pipelining of the sum blocks in the output neurons of the RVTDNN8-pip, RVTDNN10-pip, and RVTDNN15-pip can not be optimized because it remains some two cascaded adders that are not separated by a delay. The full-pipelining of these sum blocks requires additional delays that involve a total output latency of 24 delays for these models.

The proposed architectures are compared with our previously published ones [7, 8] (Table 3). For the pseudo-conventional architectures, the improvement of the MOF from 28.958 [7] to 38.635 MHz is principally due to : (i) the use of right shifter instead of multiplier to implement multiplication by the learning rate η that permits also to reduce the number of the used DSP48E1s from 256 to 240, and (ii) the use of 1-bit format to represent the 0 and 1 constant values used to compute the hyperbolic tangent and its derivative, respectively. For the pipelined architectures, the improvement of the MOF from 139.391 [8] to 212.865 MHz is essentially due to : (i) the reduction of the bit-width from 36 to 24 bits (FIX24RS instead of FIX36WT [7]), (ii) the use of

Table 1 Resource utilization and maximum operating frequency for conventional, pseudo-conventional, and pipelined architectures having 6 hidden neurons

Architecture	Conventional (RVTDNN6-con)	Pseudo-conventional (RVTDNN6-pse)		Pipelined (RVTDNN6-pip)	
	Dist. memory	Dist. memory	Block RAM	Dist. memory	Block RAM
Resource utilization					
Slices (37,680)	37,357 (99.1 %)	37,415 (99.3 %)	4,302 (11.4 %)	37,590 (99.8 %)	5,573 (14.8 %)
Flip-flops (301,440)	1,448 (0.5 %)	2,478 (0.8 %)	2,376 (0.8 %)	10,704 (3.5 %)	10,602 (3.5 %)
LUTs (150,720)	146,600 (97.3 %)	145,802 (96.7 %)	11,492 (7.6 %)	146,870 (97.4 %)	15,570 (10.3 %)
Bonded IOs (600)	145 (24.2 %)	145 (24.2 %)	145 (24.2 %)	145 (24.2 %)	145 (24.2 %)
RAMB36E1s (416)	0 (0.0 %)	0 (0.0 %)	132 (31.7 %)	0 (0.0 %)	132 (31.7 %)
DSP48E1s (768)	240 (31.2 %)	240 (31.2 %)	240 (31.2 %)	240 (31.2 %)	240 (31.2 %)
Max. Opera. Freq. (MHz)	18.703	33.820	38.635	81.280	212.865

Resource availability are given between brackets in the left column. Each Virtex-6 FPGA slice contains 4 LUTs and 8 flip-flops, only some slices can use their LUTs as distributed RAM. Each DSP48E1 slice contains a 25 $\times$ 18 multiplier, an adder, and an accumulator

Table 2 Resource utilization and maximum operating frequency for various pseudo-conventional and pipelined architectures based on block RAM memory

Architecture	Pseudo-conventional			Pipelined		
	RVTDNN8-pse	RVTDNN10-pse	RVTDNN15-pse	RVTDNN8-pip	RVTDNN10-pip	RVTDNN15-pip
Resource utilization						
Slices (37,680)	5,591 (14.8 %)	6,450 (17.1 %)	10,048 (26.7 %)	7,297 (19.4 %)	9,056 (24.0 %)	13,337 (35.4 %)
Flip-flops (301,440)	3,104 (1.0 %)	3,832 (1.3 %)	5,652 (1.9 %)	14,024 (4.6 %)	17,398 (5.8 %)	25,857 (8.6 %)
LUTs (150,720)	14,842 (9.8 %)	18,449 (12.2 %)	27,917 (18.5 %)	20,682 (13.7 %)	25,737 (17.1 %)	38,333 (25.4 %)
Bonded IOs (600)	145 (24.2 %)	145 (24.2 %)	145 (24.2 %)	145 (24.2 %)	145 (24.2 %)	145 (24.2 %)
RAMB36E1s (416)	176 (42.3 %)	220 (52.9 %)	330 (79.3 %)	176 (42.3 %)	220 (52.9 %)	330 (79.3 %)
DSP48E1s (768)	320 (41.7 %)	400 (52.1 %)	600 (78.1 %)	320 (41.7 %)	400 (52.1 %)	600 (78.1 %)
Max. Opera. Freq. (MHz)	37.474	37.453	37.564	207.555	207.555	207.555

RVTDNN8, RVTDNN10, and RVTDNN15 refer to RVTDNN networks having 8, 10, and 15 hidden neurons, respectively

right shifter instead of multiplier in the activation function, and (iii) the elimination of the saturation overflow logic in this shifter. For these architectures, the reduction of the bit-width from 36 to 24 decreases the number of the required DSP48E1s from 600 to 240 and the output latency from 24 to 16. The new pipelined architecture improves the updating delay of the synoptic weights and biases from 48 to 28 in hidden layer, and from 32 to 20 in the output layer. The number of the required RAMB36E1s is also reduced from 384 to 132 because of the size reduction of the ROM block used to approximate the activation function.

5 Experiment and results

The proposed architectures was evaluated with a 16-QAM modulated test signal generated using the communication toolbox of MATLAB/SIMULINK.

5.1 Setup

The values used to set parameters in various SIMULINK blocks are the same as those used in [7]. The connection weights and biases are randomly chosen using *randn* function of MATLAB to initialize their corresponding registers.

For an effective comparison, the connection weights of the RVTDNN networks are initialized with the same random values. Also, the same input signal is applied to all the evaluated RVTDNN architectures.

5.2 Mean square error

The mean square error (MSE) is used to compare the convergence speed of the proposed architectures. It is calculated at the RVTDNN outputs using non-overlapped frames of L samples each. For the pth analyzed frame,

Table 3 Comparison of the proposed methods to those previously published [7, 8]

Architecture	Pseudo-conventional			Pipelined	
	RVTDNN6_24_15 [7]	Pse. RVTDNN [8]	RVTDNN6-pse	Pip. RVTDNN [8]	RVTDNN6-pip
Data fixed-point format [7]	FIX24RS	FIX36TW	FIX24RS	FIX36TW	FIX24RS
Activation function					
Address bus size (NB)	15	16	15	16	15
Weight update delay					
Hidden layer (samples)	1	1	1	48	28
Output layer (samples)	1	1	1	32	20
Output latency (samples)	1	1	1	24	16
Resource utilization					
Slices (37,680)	4,082 (10.8 %)	1,755 (4.7 %)	4,302 (11.4 %)	5,732 (15.2 %)	5,573 (14.8 %)
RAMB36E1s (416)	132 (31.7 %)	384 (92.3 %)	132 (31.7 %)	384 (92.3 %)	132 (31.7 %)
DSP48E1s (768)	256 (33.3 %)	600 (78.1 %)	240 (31.2 %)	600 (78.1 %)	240 (31.2 %)
Max. Opera. Freq. (MHz)	28.958	21.327	38.635	139.391	212.865

The activation function calculated in hidden neurons ($N_1 = 6$) of each architecture is based on the block RAM memories

$$\mathrm{MSE}(p) = \frac{1}{L}\sum_{n=0}^{L-1} E(n+pL) \qquad p \geq 0 \tag{24}$$

where $E(n)$ is the total error energy [7]. At the sampling frequency of 8 MHz, a frame of $L = 800$ samples corresponds to 0.1 ms. To illustrate the convergence speed of the proposed architectures, only the first 100 frames corresponding to 10 ms are considered. As the synoptic wights and biases are randomly initialized at each test, the MSE is computed by averaging the results obtained by 10 different learning tests. Each test is performed on the first 100 frames ($0 \leq p \leq 99$) with different random initializing values.

5.3 Simulation results

Figures 11, 12, 13 and 14 show the time-domain cartesian components, constellation, AM/AM and AM/PM characteristics of the conventional, pseudo-conventional and pipelined architectures of the RVTDNN6 PA model. Note that both nonlinearity and memory effects are well modeled by the three architectures. Figure 15 shows the power spectral density (PSD) curves obtained with the three architectures, where the spectral regrowth caused by the power amplifier non-linearity is efficiently modeled. However, the pipelined architecture (RVTDNN6-pip) gives the closest curve at the bandpass borders than those obtained by the conventional (RVTDNN6-con) and pseudo-conventional (RVTDNN6-pse) ones. This difference can be explained by the saturation of the 13-bit left shifter output for the conventional and pseudo-conventional architectures but not for the pipelined one. Figure 16 shows the averaged MSE of the RVTDNN6-con, RVTDNN6-pse, and RVTDNN6-pip architectures obtained

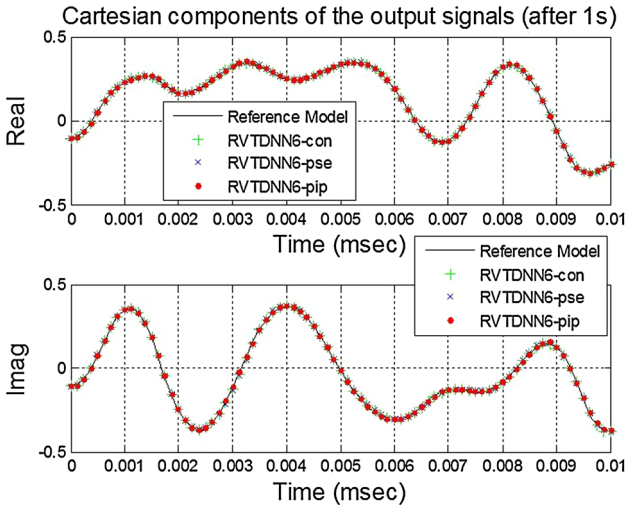

Fig. 11 Output signal in the time domain for the reference, conventional, pseudo-conventional and pipelined architectures of the RVTDNN6 model

by averaging the results (MSE) of 10 different tests. Compared over the first 100 successive frames, the RVTDNN6-pip model convergence more slowly than the two others because of its lower learning rate η and its delayed connection weights and biases update. However, the difference between the convergence times remain negligible because it does not exceed 3 ms.

Pseudo-conventional and pipelined architectures with different number of neurons in the hidden layer are also evaluated in terms of PSD and MSE curves. The PSD curves obtained by the pipelined architectures are very close to those obtained by the reference model (Fig. 17). Models with more neurons in the hidden layer give better

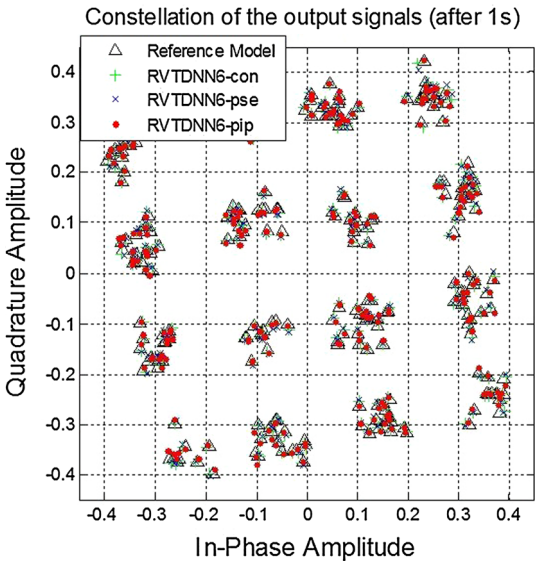

Fig. 12 Output signal constellation for the reference, conventional, pseudo-conventional and pipelined architectures of the RVTDNN6 model

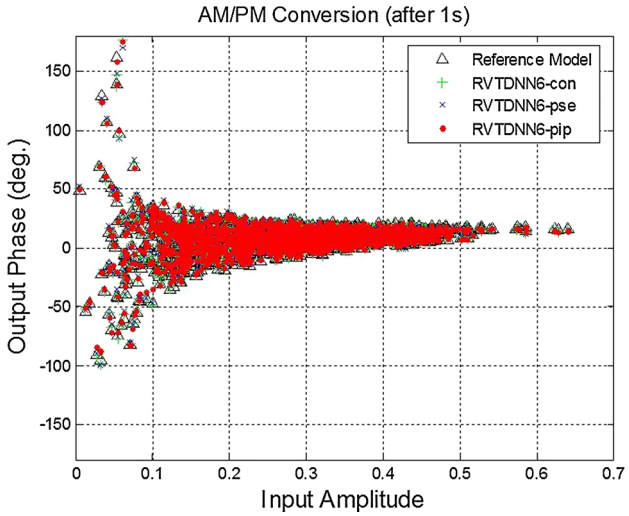

Fig. 14 AM/PM characteristics for the reference, conventional, pseudo-conventional and pipelined architectures of the RVTDNN6 model

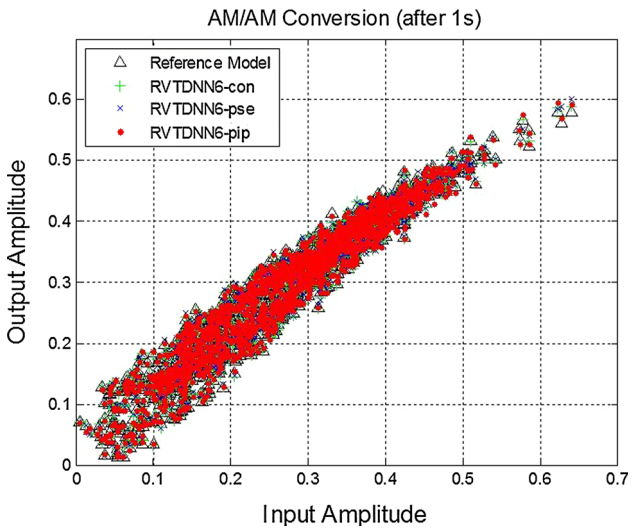

Fig. 13 AM/AM characteristics for the reference, conventional, pseudo-conventional and pipelined architectures of the RVTDNN6 model

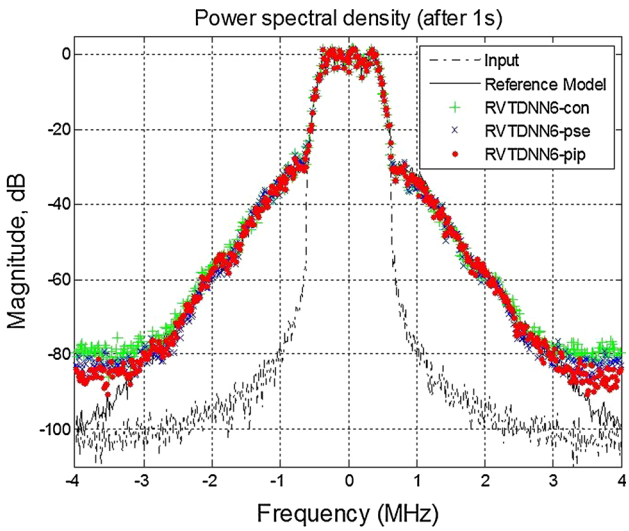

Fig. 15 Power spectral density of input and output of the reference, conventional, pseudo-conventional and pipelined architectures of the RVTDNN6 model

performance at the bandpass borders. The convergence time of these architectures is not really affected by the number of the hidden neurons (Fig. 18). Similar PSD curves are obtained by the pseudo-conventional architectures but the convergence speed (MSE) is slightly better as in Fig. 16.

Based on these simulation results, it can be seen that the proposed pipelined architecture provides similar performances than those obtained by the conventional architectures. The maximum operating frequency is considerably improved (from 18.703 to 212.865 MHz) by pipelining at the cost of the output latency of 16 delays.

5.4 Hardware co-simulation

Once the designed model is verified by SIMULINK simulation using XSG blockset, it can be executed on actual FPGA using the hardware co-simulation compilation provided by the System Generator. Depending on the targeted

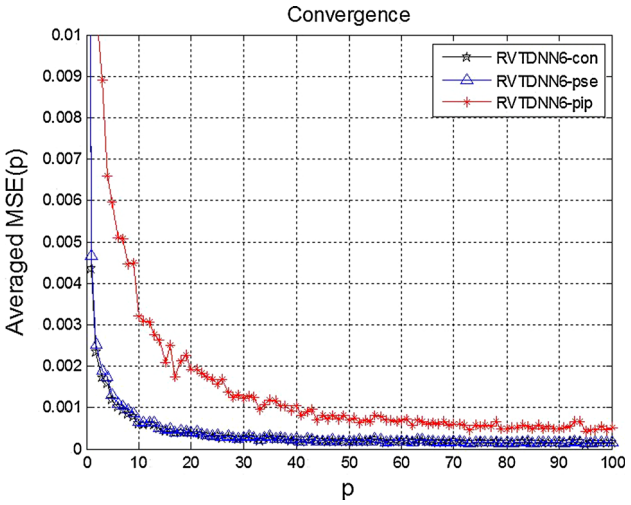

Fig. 16 Averaged mean square error (MSE) of the RVTDNN6-con, RVTDNN6-pse, and RVTDNN6-pip architectures obtained by averaging the results of 10 different tests with $L = 800$

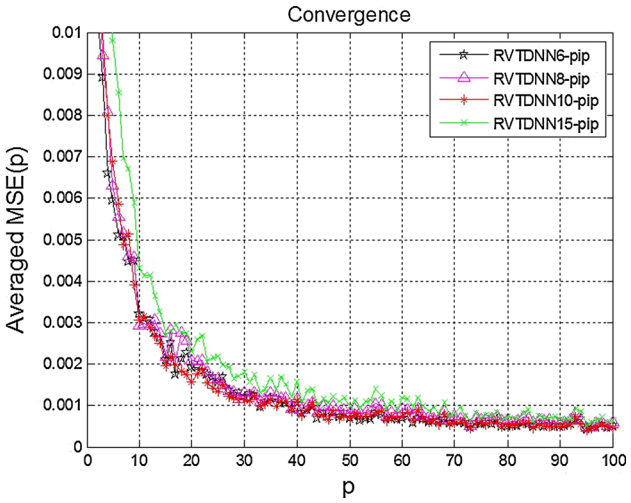

Fig. 18 As in Fig. 16 but for pipelined models with 6, 8, 10 and 15 neurons in the hidden layer

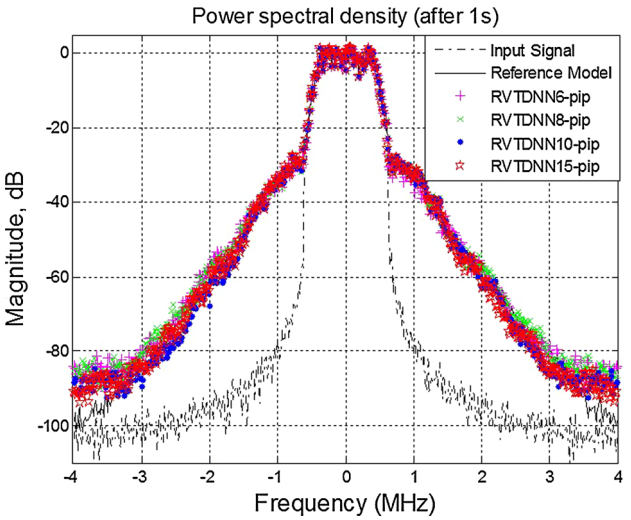

Fig. 17 As in Fig. 15 but for pipelined models with 6, 8, 10 and 15 neurons in the hidden layer

board, this compilation automatically creates a bitstream and associated it to a SIMULINK block. When the design is simulated, the compiled model is executed in hardware with the flexible simulation environment of SIMULINK. In the single-step clock mode, the FPGA is clocked from SIMULINK, whereas in the free-running clock mode, the FPGA is clocked by an internal clock. In other words, in the single-step clocking mode, the hardware co-simulation block is bit-true and cycle-true to the original model.

For the ML605 Evaluation Kit, *System Generator* block allows us to choose a clock frequency target design of 33.3, 50 or 100 MHz. The hardware co-simulation can be done only for architectures which their operating frequencies are greater than the chosen clock frequency. However, despite the fact that the maximum operating frequency of the pseudo-conventional architectures (see Tables 1, 2) is greater than 33.3 MHz, it was not possible to create the hardware co-simulation blocks for these architectures at the lowest clock frequency. Consequently, hardware co-simulation is successfully accomplished only for the pipelined architectures even if the largest clocking frequency of 100 MHz is chosen.

Figure 19 shows the top-level SIMULINK diagram that allows to compare the hardware simulation of the pipelined architecture to its SIMULINK simulation. The same input signal and the same initial values of the synoptic weight are used for the both single-step and free-running clock modes.

Figure 20, 21 and 22 present the time-domain cartesian components, constellation, and PSD of the output signal, respectively. There are obtained for the reference model and the pipelined model (RVTDNN6-pip) executed on software (SW Simulation) and hardware using both single-stepped (HW single-stepped) and free-running (HW free-running) simulation modes after 1 ms and 1 s of the time simulation. It can be noted that the hardware co-simulation with simple-stepped clock mode gives the same performances (bit-true and cycle-true) as the software simulation over all the simulation time. Also, the output signals from the hardware co-simulation with the free-running clock mode leads those from the software simulation by 16 samples. This is due to the fact that the latency of 16 periods in the hardware simulation are negligible, compared to those in the SIMULINK simulation. Consequently, the hardware simulation with the free-running clock mode converges very quickly, compared to the hardware simulation using single-step clocking mode.

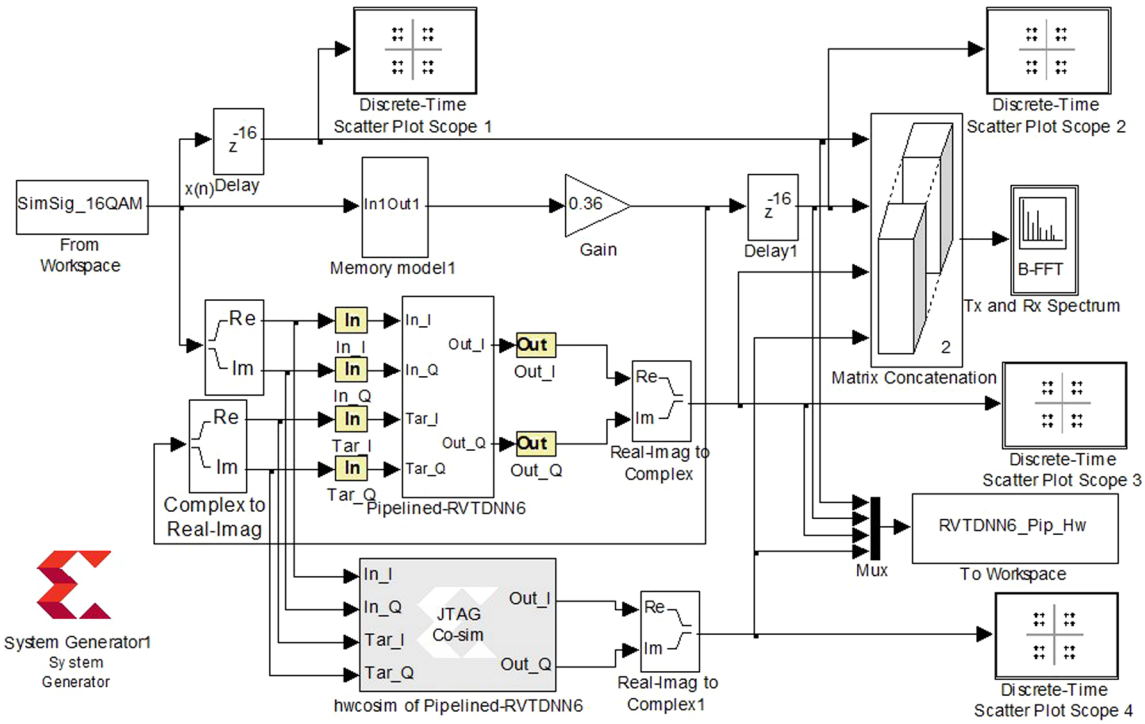

Fig. 19 Top-level SIMULINK diagram of the hardware/software cosimulation

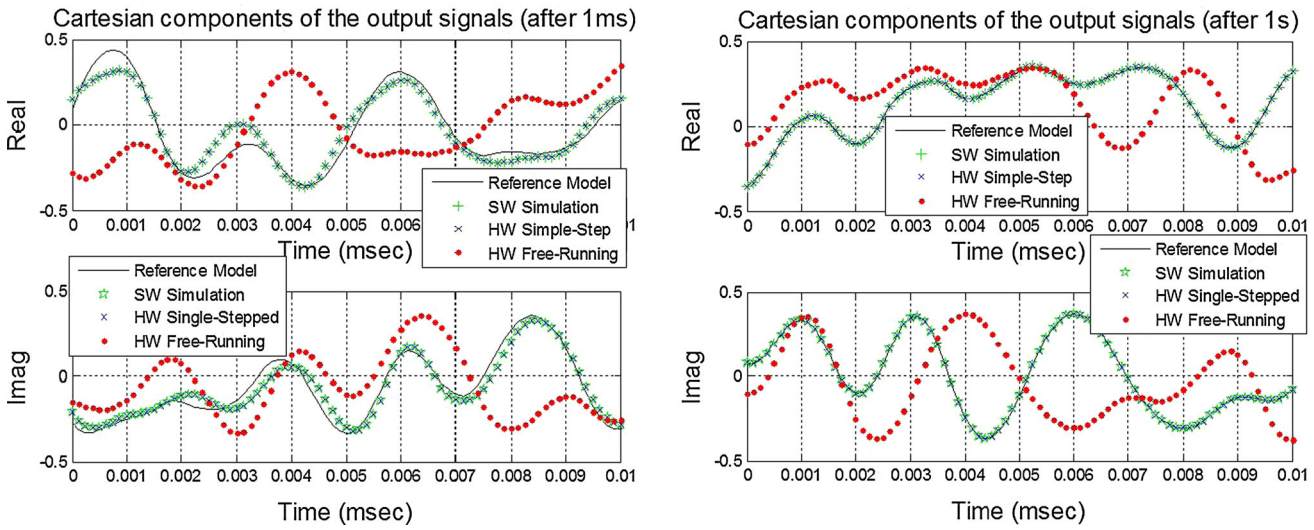

Fig. 20 Output signal waveform of the pipelined model (RVTDNN6-pip) executed on software and hardware using both single-stepped and free-running simulation modes after 1 ms (*left*) and 1 s (*right*) of SIMULINK time simulation

After only 1 ms of the time simulation, the waveform, constellation, and PSD of the signal output obtained by hardware simulation using the free-running clock mode are very close to those obtained by the reference model, which is not yet the case for the hardware simulation using the single-step clocking mode. The high-speed convergence of the hardware simulation using the free-running mode can be explained by the fact that between two events occurring

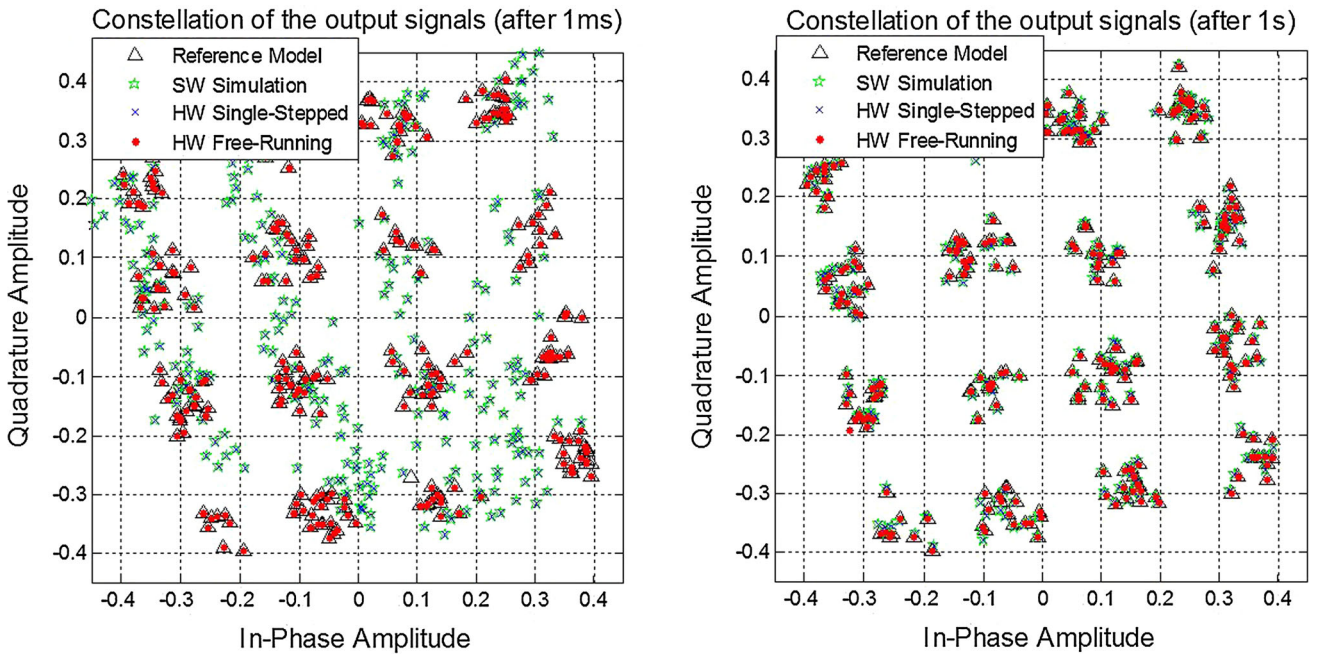

Fig. 21 Output signal constellation of the pipelined model (RVTDNN6-pip) executed on software (SW simulation) and hardware using both single-stepped (HW single-stepped) and free-running (HW free-running) simulation modes after 1 ms (*left*) and 1 s (*right*) of SIMULINK time

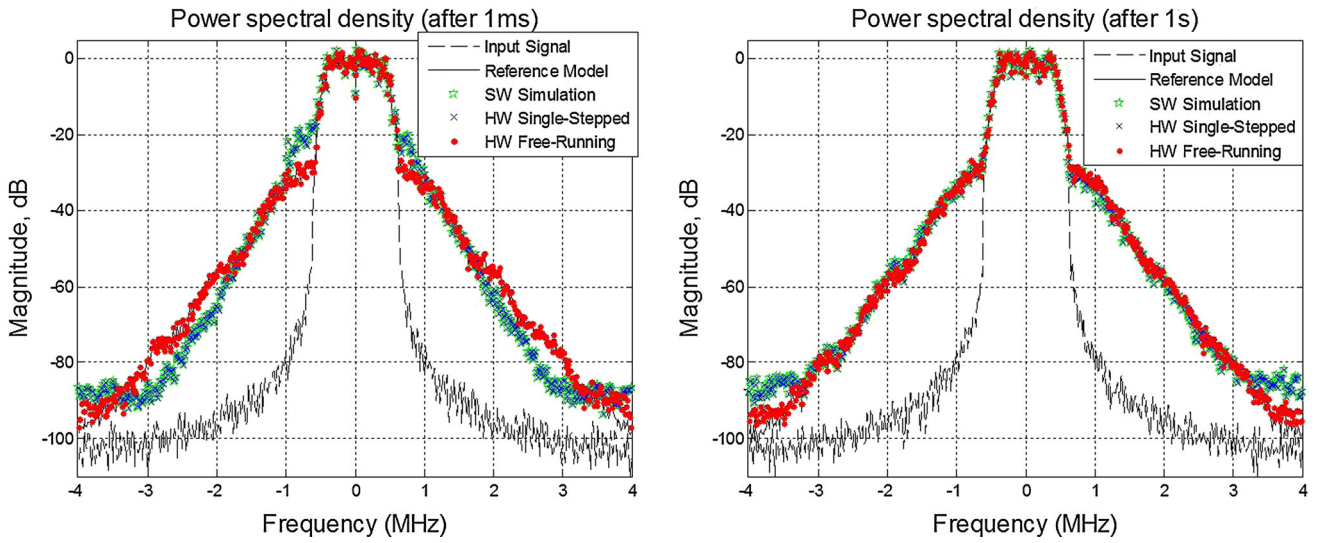

Fig. 22 Power spectral density of the pipelined model (RVTDNN6-pip) executed on software (SW simulation) and hardware using both single-stepped (HW single-stepped) and free-running (HW free-running) simulation modes after 1 ms (*left*) and 1 s (*right*) of SIMULINK time

on a SIMULINK port (one period of the processed signal), a great number of clock cycles can be elapsed in the hardware between these port events. In other words, the hardware simulation, using the free-running clock mode, can be seen as an up-sampling that improves the convergence speed of the neural network based model.

6 Conclusion

New pipelined architecture of real-valued time-delay neural network (RVTDNN) is proposed and implemented on Virtex-6 FPGA for baseband dynamic PA behavioral modeling. The proposed architecture presents higher

operating frequency and lower output latency, compared to the previously published ones. It requires less hardware resources in terms of RAMB36E1 blocks and DSP48E1 slices. The results obtained with hardware/software co-simulation using a 16-QAM modulated test signal show the high performances of this low-cost architecture. Using only 24-bit fixed-point format, the hardware-based architecture provides the same performance than that based on MATLAB software floating-point format (double precision). Moreover, the continuous update of the synoptic weights and biases on the FPGA provides a dynamic modeling that prevents any eventual change in PA's characteristics.

In the future, the proposed pipelined architecture will be used as DPD to linearize real PAs.

References

1. Liu, T., Boumaiza, S., Ghannouchi, F. M. (2004). Dynamic behavioral modeling of 3G power amplifiers using real-valued time-delay neural networks. *IEEE Transactions on Microwave Theory and Techniques, 52*(3), 1025–1033.
2. Hwangbo, H., Jung, S. C., Yang, Y., Park, C. S., Kim, B. S., Nah, W. (2006). Power amplifier linearization using an indirect-learning-based inverse TDNN model. *Microwave Journal, 49*(11), 76–88.
3. Cao, Y., Chen, X., Wang, G. (2009). Dynamic behavioral modeling of nonlinear microwave devices using real-time recurrent neural network. *IEEE Transactions on Electron Devices, 56*(5), 1020–1026.
4. Rawat, M., Rawat, K., Ghannouchi, F. M. (2010). Adaptive digital predistortion of wireless power amplifiers/transmitters using dynamic real-valued focused time-delay line neural networks. *IEEE Transactions on Microwave Theory and Techniques, 58*(1), 95–104.
5. Mkadem, F., Boumaiza, S. (2011). Physically inspired neural network model for RF power amplifier behavioral modeling and digital predistortion. *IEEE Transactions on Microwave Theory and Techniques, 59*(4), 913–923.
6. Hui, M., Liu, T., Ye, Y., Zhang, H., Shen, D., Li, L. (2012). Dynamic behavioral modeling for strongly nonlinear Doherty PAs using real-valued time-delay recurrent RBF model. *Journal of Electronics (China) ,29*(1–2), 39–45.
7. Bahoura, M., Park, C.-W. (2012). FPGA-implementation of dynamic time delay neural network for power amplifier behavioral modeling. *Analog Integrated Circuits and Signal Processing, 73*(3), 819–828.
8. Ntouné, R. S. N., Bahoura, M., Park, C.-W. (2012). FPGA-implementation of pipelined neural network for power amplifier modeling. In:*The 10th IEEE International NEWCAS Conference, Montréal, Canada* (pp. 109–112).
9. Dias, F. M., Antunes, A., Mota, A. M. (2004). Artificial neural networks: A review of commercial hardware. *Engineering Applications of Artificial Intelligence, 17*(8), 945–952.
10. Misra, J., Saha, I. (2010). Artificial neural networks in hardware: A survey of two decades of progress. *Neurocomputing ,74*, 239–255.
11. Gironés, R. G., Palero, R. C., Boluda, J. C., Cortés, A. S. (2005). FPGA implementation of a pipelined on-line backpropagation. *Journal of VLSI Signal Processing Systems, 40*, 189–213.
12. Ortigosa, E. M., Cañas, A., Ros, E., Ortigosa, P. M., Mota, S., Díaz, J. (2006). Hardware description of multi-layer perceptrons with different abstraction levels. *Microprocessors and Microsystems, 30*(7), 435–444.
13. Armato, A., Fanucci, L., Scilingo, E., Rossi, D. D. (2011). Low-error digital hardware implementation of artificial neuron activation functions and their derivative. *Microprocessors and Microsystems, 35*(6), 557–567.
14. Gilabert, P., Cesari, A., Montoro, G., Bertran, E., Dilhac, J.-M. (2008). Multi-lookup table FPGA implementation of an adaptive digital predistorter for linearizing RF power amplifiers with memory effects. *IEEE Transactions on Microwave Theory and Techniques, 56*(2), 372–384.
15. Kwan, A., Ghannouchi, F., Hammi, O., Helaoui, M., Smith, M. (2012). Look-up table-based digital predistorter implementation for field programmable gate arrays using long-term evolution signals with 60 MHz bandwidth. *Science, Measurement Technology, IET, 6*(3), 181–188.
16. Gilabert, P. L., Montoro, G. (2013). Computationally efficient real-time digital predistortion architectures for envelope tracking power amplifiers. *International Journal of Microwave and Wireless Technologies, 5*, 187–193.
17. Ku, H., Kenney, J. S. (2003). Behavioral modeling of nonlinear RF power amplifiers considering memory effects. *IEEE Transactions on Microwave Theory and Techniques, 51*(12), 2495–2504.
18. Ding, L., Zhou, G. T., Morgan, D. R., Ma, Z., Kenney, J. S., Kim, J., Giardina, C. R. (2004). A robust digital baseband predistorter constructed using memory polynomials. *IEEE Transactions on Communications, 52*(1), 159–165.
19. Mrabet, N., Mohammad, I., Mkadem, F., Rebai, C., Boumaiza, S. (2012). Optimized hardware for polynomial digital predistortion system implementation. In: *2012 IEEE Topical Conference on Power Amplifiers for Wireless and Radio Applications (PAWR)*.
20. Guan, L., Zhu, A. (2010). Low-cost FPGA implementation of Volterra series-based digital Predistorter for RF power amplifiers. *IEEE Transactions on Microwave Theory and Techniques, 58*(4), 866–872.
21. Zhu, A., Pedro, J., Brazil, T. (2006). Dynamic deviation reduction-based Volterra behavioral modeling of RF power amplifiers. *IEEE Transactions on Microwave Theory and Techniques, 54*(12), 4323–4332.
22. Liszewski, J., Schubert, B., Keusgen, W., Kortke, A. (2011). Low-complexity FPGA implementation of Volterra predistorters for power amplifiers. In: *The IEEE Topical Conference on Power Amplifiers for Wireless and Radio Applications (PAWR)* (pp. 41–44).
23. Jiménez, V., Jabrane, Y., Armada, A., Said, B., Ouahman, A. (2011). High power amplifier pre-distorter based on neural-fuzzy systems for OFDM signals. *IEEE Transactions on Broadcasting, 57*(1), 149–158.
24. Kwan, A., Helaoui, M., Boumaiza, S., Smith, M., Ghannouchi, F. (2009). Wireless communications transmitter performance enhancement using advanced signal processing algorithms running in a hybrid DSP/FPGA platform. *Journal of Signal Processing Systems, 56*(2–3), 187–198.
25. Haykin, S. (1999). *Neural networks: A comprehensive foundation* (2^{nd} ed.). Upper Saddle River: Prentice-Hall.
26. Bahoura, M., Ezzaidi, H. (2011). FPGA-implementation of parallel and sequential architectures for adaptive noise cancelation. *Circuits, Systems, and Signal Processing, 30*(6), 1521–1548.
27. Bahoura, M., Park, C.-W. (2011). FPGA-implementation of high-speed MLP neural network. In: *The 18th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Beirut, Lebanon* (pp. 426–429).

28. Bahoura, M., Ezzaidi, H. (2012). FPGA-implementation of discrete wavelet transform with application to signal denoising. *Systems, and Signal Processing, 31*(3), 987–1015.
29. Long, G., Ling, F., Proakis, J. (1989). LMS algorithm with delayed coefficient adaptation. *IEEE Transactions on Acoustics, Speech, and Signal Processing, 37*(9), 1397–1405.
30. Saleh, A. A. M. (1981). Frequency-independent and frequency-dependent nonlinear models of TWT amplifiers. *IEEE Transactions on communications, 29*(11), 1715–1720.

Mohammed Bahoura received the B.Sc. degree in Electronic Engineering from the Université des Sciences et de la Technologie of Algiers, Algeria, in 1990, the M.Sc. degree in Instrumentation and Control from the Université de Rouen, France, in 1994 and the Ph.D. degree in Electrical Engineering, from the same University, in 1999. He was a Postdoctoral Fellow at the Université du Québec à Chicoutimi, Canada, from 1999 to 2001. Since 2001, he has been with the Université du Québec à Rimouski, where he is a Professor of Electrical Engineering. His research interests focus on digital signal processing, biomedical engineering, speech enhancement, underwater acoustic, communication systems, and hardware implementation on FPGAs.