Digital Circuit Realization of Piecewise-Affine Functions With Nonuniform Resolution: Theory and FPGA Implementation

Tomaso Poggi, Francesco Comaschi, and Marco Storace, Member, IEEE

Abstract—This brief proposes a digital circuit architecture implementing a class of continuous piecewise-affine (PWA) functions. The work rests on a previous architecture realizing PWA functions with uniform resolution. By using PWA mapping that can be implemented through a few simple functional blocks, it is possible to extend the representation capabilities of the architecture to PWA functions with nonuniform resolution. After defining the mapping and the corresponding functional blocks, the proposed architecture is implemented in a field-programmable gate array, and a simple example is shown.

Index Terms—Digital architectures, nonlinear circuits, nonuniform resolution, piecewise-affine (PWA) functions.

I. INTRODUCTION

PIECEWISE-LINEAR or piecewise-affine (PWA) multivalued functions are the core of many recent and less recent works in the circuit and system [1]–[7], control [8], [9], fuzzy system [10]–[13], and neural network [14]–[16] communities. Such an interest in PWA functions can surely be ascribed to their conceptual simplicity, which allows one to exploit the large body of knowledge provided by linear system theory, and to the very good accuracy in the representation of nonlinear functions, even multivalued, achievable through approximation/interpolation with sufficiently dense sampling [17].

Basically, there are two main research lines (often strictly related) concerning PWA functions: modeling (i.e., identification or approximation) and implementation of nonlinear systems. This brief is concerned with the digital circuit implementation of PWA functions. Then, we do not focus on modeling problems, and we assume to have a PWA function to implement. In most cases, this function is the result of a preliminary modeling step providing the optimal PWA function for a given approximation problem. This step is usually taken offline by using a computer, whereas we deal with the online computation of the PWA function by using dedicated digital circuits.

In many cases, PWA functions are implemented (often through interpolators) on computers or digital signal processor boards. When embedded real-time small-size low-power non-linear (not necessarily PWA) function evaluation is required, the circuit implementation of PWA functions might be of interest by resorting to either programmable hardware such as field-programmable gate arrays (FPGAs) or even dedicated integrated circuits.

In this context, many architectures have been proposed in recent years for the implementation of PWA functions [11], [12], [18], [19]. For a review, see [20]. Most of the architectures proposed so far are based on a uniform partition (called simplicial partition) of the domain of the PWA function to be implemented. The resulting PWA function is linear over each simplex and is completely defined by the values it takes at the vertices of the partition. The main limit of such an approach (called uniform-resolution approach) is that the implementable functions are defined with uniform resolution over the whole domain since all the simplices are identical.

Quite often, one needs to implement a PWA function with different resolution requirements in different domain subregions. In [21], a method has been proposed to obtain a multiresolution PWA approximation of a given function. This method is efficient from a computational point of view but not suitable for an efficient circuit implementation.

In this brief, we use a different approach, based on invertible mapping between uniformly and nonuniformly partitioned domains, that allows one to implement PWA functions with nonuniform resolution. We show that there is a simple and efficient way to implement nonuniform-resolution PWA functions by extending the results reported in [19] and [22]. The FPGA implementation of a 3-D PWA function is proposed and discussed.

II. CIRCUIT IMPLEMENTATION OF UNIFORM-RESOLUTION PWA FUNCTIONS

Here, we briefly summarize the elements of the uniform resolution approach that are essential to introduce the nonuniform resolution one. For a deeper treatment of this topic, the reader is referred to [22]. We deal with a continuous PWA function \( f_{\text{PWA}} : \mathbb{R} \rightarrow \mathbb{R} \), defined over a properly scaled \( n \)-dimensional compact domain \( S_z = \{ z \in \mathbb{R}^n : 0 \leq z_i \leq m_i, i = 1, \ldots, n, m_i \in \mathbb{N} \} \). \( f_{\text{PWA}} \) can be easily circuit-implemented by introducing a regular partition of the domain \( S_z \) [22]: Each dimensional component \( z_i \) of the domain \( S_z \) is divided into \( m_i \) subintervals of unitary length. As a consequence, the domain \( S_z \) is partitioned into \( \prod_{i=1}^n (m_i + 1) \) hypersquares and contains \( N = \prod_{i=1}^n (m_i + 1) \) vertices \( v^z_k \) collected in a set \( V_z \).
Each hypersquare can be further partitioned (uniform simplicial partition) into \( n! \) nonoverlapping hypertriangular (triangular, if \( n = 2 \)) identical regions called simplices.

The shape of a given PWA function \( f_{\text{PWA}} \) is coded by the \( N \) coefficients \( c_k \), which are the values of \( f_{\text{PWA}} \) at the vertices \( v_k \) of its simplicial domain. We assume that the coefficients are already available.

The value of \( f_{\text{PWA}}(z) \) can be calculated as a linear interpolation of the \( f_{\text{PWA}} \) values at the vertices of the simplex containing \( z \). This is as a linear interpolation of a subset of \( n + 1 \) coefficients \( c_j \), i.e.,

\[
f_{\text{PWA}}(z) = \sum_{j=0}^{n} \mu_j^z c_j \tag{1}
\]

where the \( \mu_j^z \)'s are the weights that give \( z \) as a convex combination of the vertices of the simplex that contains it, \( z = \sum_{j=0}^{n} \mu_j^z v_j \), with \( \sum_{j=0}^{n} \mu_j^z = 1 \). The coefficients \( c_j \) correspond to the vertices surrounding \( z \) \([22]\) and must be chosen according to \( z \). Every interpolation weight \( \mu_j^z \) depends, in turn, on \( z \), but these dependencies are omitted for ease of notation. As a consequence, the circuit realization of a uniform-resolution PWA function contains three functional elements:

1) a memory where the \( N \) \( c_k \) coefficients are stored;
2) a block that finds, for any given input \( z \), the coefficients \( c_j \) and \( \mu_j^z \);  
3) a block performing the weighted sum \((1)\).

III. PWA FUNCTIONS OVER NONUNIFORM PARTITIONS

We aim to define invertible mapping between a uniformly partitioned domain \( S_z \) and a nonuniformly partitioned one \( S_x \), as shown in Fig. 1 for a 2-D domain. This would allow us to exploit the uniform-resolution architecture (with minor changes) to implement PWA functions with nonuniform resolution.

A. Nonuniform Partition

Let us consider a domain \( S_x = [0, 1]^n \) partitioned into \( \prod_{i=1}^{n} m_i \) nonidentical hyperrectangles.\(^1\) Then, \( S_x \) contains \( N = \prod_{i=1}^{n} (m_i + 1) \) vertices \( v_k \) collected into the set \( V_x \). Each element in \( V_x \) has rational coordinates \( v_k = (x_1^{k_1}, x_2^{k_2}, \ldots, x_n^{k_n})^T \) (where \( k_i \in \{0, \ldots, m_i\}, i = 1, \ldots, n \), and the length of each subinterval is a positive power of two, i.e., \( x_i^{k_i+1} - x_i^{k_i} = 2^{-q_i} \), where \( q_i \) is a positive integer that can be different for each subinterval. For the coordinates of the vertices \( v_k \), the subscript index defines the domain component, and the superscript index establishes an increasing order for the coordinates' components (see Fig. 1).

On the other hand, \( S_z \) contains \( N \) vertices \( v_k \) with natural coordinates \( v_k = (k_1, k_2, \ldots, k_n)^T \), where \( k_i \in \{0, \ldots, m_i\}, i = 1, \ldots, n \); the length of each subinterval is unitary. Every dimensional component of domains \( S_x \) and \( S_z \) is divided into \( m_i \) subintervals. We can also define a nonuniform simplicial partition for \( S_z \) by subdividing each hyperrectangle into \( n! \) nonoverlapping hypertriangular regions and express \( f_{\text{PWA}} : S_x \to \mathbb{R} \) as a weighted sum

\[
f_{\text{PWA}}(x) = \sum_{j=0}^{n} \mu_j^z c_j \tag{2}
\]

where \( \{\mu_j^z\}'s \) are the weights of the convex combination \( x = \sum_{j=0}^{n} \mu_j^z v_j \).

B. Mapping \( T \)

We define the hypersquare \( R_k^z \) of \( S_z \) as the following Cartesian product:

\[
R_k^z = [k_1, k_1 + 1] \times [k_2, k_2 + 1] \times \cdots \times [k_n, k_n + 1].
\]

We point out that the (unique) vertex of \( R_k^z \) closest to the origin is \( v_k \) (see Fig. 1).

Analogously, the hyperrectangle \( R_k^x \) of \( S_x \) is given by

\[
R_k^x = [x_1^{k_1}, x_1^{k_1+1}] \times [x_2^{k_2}, x_2^{k_2+1}] \times \cdots \times [x_n^{k_n}, x_n^{k_n+1}].
\]

Also, in this case, the (unique) vertex of \( R_k^x \) closest to the origin is \( x_k \).

We aim to map each hyperrectangle \( R_k^x \) into the hypersquare \( R_k^z \) for any \( k \), as shown in Fig. 1 for a generic \( k \).

The mapping \( T(x) \) is PWA over \( S_x \) and has the form \( T(x) = A_k x + b_k, x \in R_k^x \), with \( A_k \) diagonal (and invertible).

We build each component \( T_i \) of \( T \) by mapping every interval between two vertices’ coordinates along the \( i \)th axis in \( S_x \) \( \{(x_i^{k_i}, x_i^{k_i+1})\} \) into the corresponding interval along the \( i \)th axis in \( S_z \) \( \{(k_i, k_i + 1)\} \), as shown in Fig. 2. Thus, given a point \( x = (x_1, \ldots, x_n)^T \) belonging to a given hyperrectangle of \( S_x \) and a corresponding point \( z = (z_1, \ldots, z_n)^T \) belonging to the mapped hypersquare of \( S_z \), we find \( T \) by imposing, for any \( i \), the following constraints:

\[
\frac{x_i - x_i^{k_i}}{x_i^{k_i+1} - x_i^{k_i}} = \frac{z_i - k_i}{k_i + 1 - k_i}.
\]

Now, by reordering and taking into account that \( x_i^{k_i+1} - x_i^{k_i} = 2^{-q_i} \), for any \( k_i \), we have

\[
z_i = T_i(x_i) = 2^{k_i} \left(x_i - x_i^{k_i}\right) + k_i, \quad x_i \in [x_i^{k_i}, x_i^{k_i+1}].
\]

Fig. 1. Two-dimensional example of the (left) original and (right) transformed domain.

---

\(^1\) The more general case, where \( S_x = \{x \in \mathbb{R}^n : a_i \leq x_i \leq b_i, i = 1, \ldots, n\} \), can always be reported to this particular case by proper linear affine scaling of \( x \).

---

Author(s): [Insert Author(s) Here]

---

IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS, VOL. 57, NO. 2, FEBRUARY 2010

Authorized licensed use limited to: Universita di Genova. Downloaded on March 02,2010 at 03:22:28 EST from IEEE Xplore. Restrictions apply.
The circuit is made up of two parts: The first one (dark-gray blocks in Fig. 3) is completely combinatorial and implements the mapping \( T \); the second one (light-gray blocks) evaluates the PWA function defined over a uniform partition. We focus our attention on the description of the mapping block, whereas for details on the possible choices for the rest of the architecture, the reader is referred to [20].

The components of the input vector \( x \), represented with \( p \) bits each, are fed into the mapping block and processed in a parallel fashion by \( n \) subblocks \( T_i \), as shown in Fig. 3. The structure of each subblock is displayed in Fig. 4.

The terms \( q_i^k \) and \( x_i^k \) are stored in a memory and are uniquely labeled by \( k_i \). Then, the correct pair \( q_i^k \), \( x_i^k \) is retrieved by addressing the memory with the binary version of \( k_i \), i.e., a \( r \)-bit string where \( r \) is the minimum number of bits required to represent the greatest \( k_i \), \( i \in \{1, \ldots, n\} \). Given an input vector \( x \), such a string is computed by using a bank of comparators and an adder since \( k_i = \sum_{j=1}^{n} s_j \), where \( s_j = 1 \) if \( x_i \geq x_i^j \), and \( s_j = 0 \) otherwise.

Once the data are retrieved from the memory, the subtraction \( x_i - x_i^k \) is performed, and the result is shifted on the left by \( q_i^k \) bits, thus obtaining a \( p \)-bit string that is the binary version of the decimal part of \( z_i \). Owing to remark 1 in Section III-B, it is not necessary to obtain \( z_i \) by adding its integer part \( k_i \) [see (5)]; it is sufficient to concatenate the integer and decimal parts, thus obtaining a string of \( r + p \) bits. The left shift introduces \( q_i^k \) zeros in the least significant part of the decimal part of \( z_i \).

Then, the last \( t \) bits of each concatenated string (all 0) can be discarded, where

\[
t = \min_i \left\{ \min_{k_i} q_i^k \right\}
\]

and the output of the mapping block is a set of \( n \) strings of \( r + p - t \) bits.
We point out that the circuit can implement any continuous function \( f_{\text{PWA}} \) with nonuniform resolution, provided that the constraint \( x_i^{k_i+1} - x_i^{k_i} = 2^{-q_i} \) is fulfilled.

V. IMPLEMENTATION AND RESULTS

To implement the described circuit architecture on an FPGA, we used a very high speed integrated circuit hardware description language description. The code is fully generic, i.e., it is possible to change every parameter before programming the FPGA without editing the code.

The architecture has been implemented using Xilinx ISE 10.1 software on a Xilinx Spartan 3 FPGA (xc3s200-5ftp256), employing \( p = 16 \) bits to code each component of the input vector and 12 bits to code the value of the output function.

The standard part of the architecture (evaluation of the PWA function defined over a uniform partition) has been implemented by using an architecture that requires \( n + 3 \) clock cycles to completely process an input. In the first clock cycle, the circuit calculates the weights \( \mu_j^i \) and the memory addresses of the coefficients \( c_k \) (see architecture B in [20]). Then, in the following \( n + 2 \) clock cycles, sum (2) is performed using a multiply-and-accumulate block.

We tested the circuit implementing a PWA version of the function

\[
f(x_1, x_2, x_3) = e^{-3x_1} (x_2 + x_3^2) + e^{-100((x_1-0.8)^2+(x_2-0.7)^2)},
\]

(7)

The PWA approximation \( f_{\text{PWA}} \) of this function has been obtained by using standard optimization techniques (least squares) and is out of the scope of this brief. We point out that finding the partition and the approximation coefficients is an offline process, whereas in this brief, we focus on the online computation of a known PWA function.

Functions (7) and \( f_{\text{PWA}} \) are defined over the domain \([0,1]^3\), partitioned in the following way:

\[
\begin{align*}
p_1 &= (0, 0.25, 0.5, 0.625, 0.75, 0.875, 1) \\
p_2 &= (0, 0.5, 0.625, 0.75, 0.875, 1) \\
p_3 &= (0, 0.5, 0.75, 1)
\end{align*}
\]

where \( p_i = (x_1^i, x_2^i, \ldots, x_3^{m_i}) \) collects the coordinates of the vertices along each dimension (\( m_1 = 6, m_2 = 5, m_3 = 3 \)). In this case, the number of coefficients \( c_k \) to be stored in the memory is \( N = 168 \), while \( r = 3 \) and \( t = 1 \). The estimated maximum working frequency is 206 MHz, which corresponds to a throughput of one sample every 34.3 ns, with a power consumption of 46 mW. For higher frequencies, a correct behavior of this circuit is not guaranteed. Anyway, we did not optimize the FPGA implementation, then further improvements are possible.

We performed a post place-and-route simulation using ModelSim XE III evaluating the function over a grid of 27,000 points uniformly distributed over the domain. The output values have been compared with a MATLAB (double-precision) implementation of (7). The obtained relative error is

\[
e = \frac{\sum_{j=1}^{27000} |f(x^j) - f_{\text{FPGA}}(x^j)|}{\sum_{j=1}^{27000} |f(x^j)|} = 0.17
\]

where \( f_{\text{FPGA}} \) is the function evaluated by the FPGA, and \( x^j \) are the samples.

Of course, since we are not interested in the modeling aspects of the problem, this number is not significant per se. However, to show the benefits of the nonuniform partition in a case like this one, we can determine how many vertices would be required to obtain the same relative error by using a PWA function with a uniform partition: Generally speaking, the more the vertices, the higher the circuit complexity. To this end, we first approximated \( f \) using a uniform partition with the same number (\( N = 168 \)) of vertices, thus obtaining a higher relative error (\( e = 0.26 \)). To implement \( f \) with the same accuracy as for the nonuniform case, the number of vertices in the uniform partition must be increased. For instance, taking \( m_1 = 8, m_2 = 8, \) and \( m_3 = 3 \) subdivisions along the domain axis gives an error \( e = 0.16 \), but the number of coefficients stored in the memory is almost doubled (\( N = 324 \)).

After the tests carried out with ModelSim, we measured some signals through an HP16600 logic state analyzer to test the real implementation. Fig. 5 shows a snapshot for an input clock frequency of 20 MHz. The frequency of the clock signal chosen to perform the measurements is much lower than the maximum allowed working frequency (206 MHz) due to the used instrumental setup. For a standard use, one can mount the FPGA on a printed circuit board and might choose higher clock frequencies (up to about 206 MHz).

When the ready signal goes down, the circuit acquires a new input vector on the next falling edge of the clock \( CK \). The transformation block is completely combinatorial; thus, the stored value is already scaled in the domain \( S_i \). Then, the multiply-and-accumulate block provides sum (2), and the output value is forwarded to an output register when ready goes high again.

VI. CONCLUSION

We have shown that an architecture realizing uniform-resolution PWA functions can be used to implement nonuniform-resolution PWA functions by preliminary adding a coordinate transformation. This operation is performed by a block that requires just a few adders and shift registers, whose number grows linearly with the dimension \( n \) of the domain. Even if the proposed architecture relies on the circuit proposed in [20], the same idea could be applied to other circuit implementations based on a uniform partition (e.g., [12]).

All the architectures implementing uniform-resolution PWA functions proposed up to now have a main common drawback: The number of coefficients (i.e., the size of the memory) needed to store a given PWA function in the circuit grows exponentially with \( n \). As shown by the example in Section V, nonuniform resolution can be thought of as a way to lower the model complexity (i.e., the number of coefficients), although maintaining the same accuracy. In particular, the implementation proposed in this brief allows one to exploit this advantage keeping the circuit simple.

Work in progress is concerned with the modeling aspects neglected in this brief. We are developing an algorithm to find a nonuniform partition that, for a given nonlinear function
to be approximated and a given set \( \{m_i\} \) (i.e., for a given circuit complexity), finds a set \( \{p_i\} \) and the corresponding PWA function ensuring the lowest approximation error according to a given metric.

**ACKNOWLEDGMENT**

The authors would like to thank D. Linaro for the helpful discussions.

**REFERENCES**


