# ON KRONECKER PRODUCTS SIGNAL PROCESSING ALGORITHMS AND FPGA COMPUTATIONAL STRUCTURES

Yvonne Avilés

Advisor: Domingo Rodríguez

University of Puerto Rico - Mayagüez Campus
Department of Electrical and Computer Engineering
**C**omputational **S**ignal **P**rocessing **G**roup (cspg@ece.uprm.edu)
Mayagüez, Puerto Rico, 00681-9042

## ABSTRACT

An environment is developed where Kronecker-based signal processing algorithms are translated into equivalent hardware constructs in Field Programmable Gate Array (FPGA) units.

This work focuses on the methodology that establishes the correspondence between Kronecker–based signal processing algorithms and hardware computational structures. Our work also presents the methodology as an alternative technique for modular design of computational structures for signal processing algorithms as well as an efficient framework for rapid prototyping. The work describes the methodology in terms of fast Fourier transform (FFT) algorithms; but, basic principles can be abstracted and generalized.

## 1. INTRODUCTION

Traditionally, signal processing tasks have been implemented using either digital signal processors (DSPs) or custom circuitry. Field Programmable Gate Arrays (FPGAs) present an alternative technology, which combines flexibility of programmable solutions and improved performance of dedicated hardware [4]. FPGA architectures comprise reconfigurable logic in an inherent parallel structure by means of an internal matrix structure of Configurable Logic Blocks (CLBs) that perform any logic function of its inputs and may be reconfigured within the system. These features provide a framework for rapid prototyping, well suited for implementation of computational structures of signal processing algorithms in FPGA.

Modularity, scalability, and reconfigurability are important factors in our conception of computational structure design. Signal processing algorithms, analyzed as stages of computation in a hardware transformation environment, provide a basis for modular design as hardware constructs in FPGAs. By this means, suitable implementations are achieved based on performance assessment.

## 2. KRONECKER PRODUCTS ALGEBRA AND HARDWARE COMPUTATIONAL STRUCTURES

A means for implementing signal processing algorithms as FPGA computational structures centers on identifying a correspondence between the functional primitives or basic factors in a mathematical composition of a given signal processing algorithm and suitable hardware constructs in an FPGA computational structure. It is this correspondence that allows us to take advantage of the properties of Kronecker products algebra, a branch of finite dimensional, multi-linear algebra, as a language tool to enhance the mapping process resulting in rapid prototyping operations [3].

Properties of Kronecker algebra provide mechanisms to express (signal processing) algorithms as compositions of factors representing equivalent stages of computation in a hardware transformation environment representing hardware functional primitives [2]. Algorithms may be expressed as distinct organizations of computational kernels, addressing then a range of computational structure implementations. This organizational property highlights reconfiguring features, which allow rapid prototyping of computational structures based on the selection and organization of the basic building blocks identified in a given algorithm.

### 2.2 Kronecker product of two matrices

Let any two arbitrary matrices **A** and **B** be of order $N \leftrightarrow N$ and $M \leftrightarrow M$, respectively. The Kronecker product of the two matrices is a new matrix, denoted by $(A \quad B)$, of order $NM \leftrightarrow NM$, given by:

$$C = A \quad B = [a_{N,N}]B$$

$$A \quad B = \begin{matrix} a_{0,0}B & a_{0,1}B & \cdots & a_{0,N-1}B \\ a_{1,0}B & a_{1,1}B & \cdots & a_{1,N-1}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{N-1,0}B & a_{N-1,1}B & \cdots & a_{N-1,N-1}B \end{matrix}$$

Matrices of this type are called *Kronecker products.* An $NM \leftrightarrow NM$ Kronecker product $A \quad B$ can be decomposed as follows:

$$A \quad B = (AI_M) \quad (I_N B) = (A \quad I_N)(I_M \quad B)$$

where $I_M$ is the $M$ - dimensional identity matrix. The previous product formulation can be analyzed in separate terms of factors called functional primitives.

$$A \quad I_N = \begin{matrix} a_{0,0} I_N & \cdots & a_{0,M-1} I_N \\ \vdots & \ddots & \vdots \\ a_{M-1,0} I_N & \cdots & a_{M-1,M-1} I_N \end{matrix}$$

$$(I_M \quad B) = \begin{matrix} B & & \\ & \ddots & \\ & & B \end{matrix}$$

The factor $(A \quad I_N)$ or *vector Kronecker factor* represents a vector operation on vectors of length $N$. On the other hand, the factor $(I_M \quad B)$ is the direct sum of $M$ copies of $B$. The action of $(I_M \quad B)$ on a vector is also known as the *parallel Kronecker product factor operation.* Accordingly, the computation of a Kronecker operation on an input $\mathbf{x}$ vector represented by $\mathbf{y} = (A \quad B)\mathbf{x}$ can be executed in two stages: One performing a vector operation and another performing a parallel operation. Each factor associated, to a distinct computational construct, establishes then equivalence between Kronecker product compositions and computational structures.

### 2.3 Kronecker–based DSP Algorithms

To implement a signal-processing algorithm using Kronecker algebra requires first that the algorithm be expressed as a Kronecker product compositions [3]. Each Kronecker factor may then be decomposed into Kronecker products of lower order until primitive Kronecker factors are identified. The resulting expression is then organized by means of Kronecker algebra properties to address vector, parallel, or mixed structures, to be implemented as hardware constructs. Selection of a Kronecker product formulation with the best performance characteristics depends on the organization structure of the target hardware.

#### *2.3.1 Kronecker decomposition of the FFT*

The general Kronecker decomposition of the FFT algorithm using the Cooley-Tukey method for decimation in time, for a Fourier matrix $F_N$ of order $N = R?S$, was presented in [5] and is shown below.

$$F_N \mathbf{x} = (F_R \quad I_S) T_{N,S} (I_R \quad F_S) P_{N,S} \mathbf{x}$$

As a fist step in describing our methodology for mapping FFT algorithms to FPGA computational structures, we present a Kronecker formulation of a Fourier matrix of order four as a Cooley-Tukey FFT decimation in time algorithm. We selected this example since it contains the smallest order Fourier computational elements which are $F_2$:

$$F_4 \mathbf{x} = (F_2 \quad I_2) T_{4,2} (I_2 \quad F_2) P_{4,2} \mathbf{x}$$

Each factor in the formulation represents a stage in the FFT algorithm to which we associate a basic hardware element or construct in the computational structure. For high order FFTs, the method is applied recursively until functional primitive factors are reached. Combination and organization of these functional modules provide a framework for designing computational structures in FPGAs.

## 3. FPGA COMPUTATIONAL STRUCTURES

As stated above, signal processing algorithms, efficiently modeled by means of Kronecker products formulations, may be directly implemented as hardware constructs, establishing a direct correspondence between hardware and software structures. Using the application example discussed earlier a computational structure for an 8-point FFT may be represented as the structure in Figure 1.

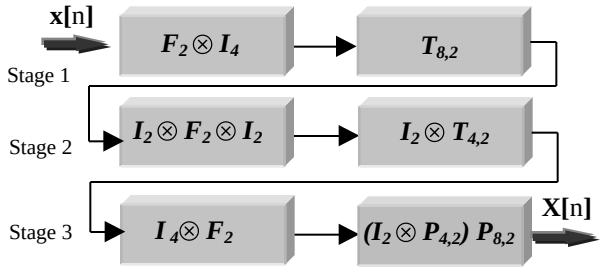


**Figure 1**: $F_8$ Computational Structure

The block elements within the computational structure represent the functional primitives.

### 3.1 Design Tools

The Xilinx Foundation Series Software with Foundation Express to be used in this work, is a complete design package comprised of design entry tools and implementation tools including a Synthesis Software from Synopsys for the VHDL hardware description language (HDL). The design entry tool includes editors for HDL, finite state machines (FSM) and schematic formats. The implementation tools perform translation, mapping, placement, route and download of FPGA designs.

### 3.2 Design Methodology

The design methodology for the Kronecker-SPA environment is defined as a procedure of two major components: *the Hardware Computational Structure (HCS) Generator Framework* and the *Hardware Computational Structure Generator Environment*, discussed below.

***HCS Generator Framework***

- ***Modeling and simulation of Kronecker-based DSP functional primitives through hardware description language.***

- ***Implementation of the basic DSP functional primitives as hardware constructs.***
  Functional primitives, as the two-point FFT(F(2), stride permutations, twiddle factor matrix operator, and other primitive functions are implemented as hardware constructs by means of hardware description language and Xilinx FPGA implementation tools.

- ***Functional Primitive Library (FPL)***
  The hardware constructs implemented in the previous stage are stored as a library of functional primitives.

- ***Design and implementation of a Scalable Operator.***
  The Scalable Operator builds large modules using functional primitives or low order constructs.

- ***Design and implementation of a Multiplication Operator (MO)***
  This operator connects any given number of constructs within the FPL to form a HCS.

***HCS Generator Environment***

Within a graphical user interface the following stages are integrated:

- ***Acquisition of SPA formulation***
  Users may create a custom SPA formulation using functional primitive constructs within the FPL tool, or select an algorithm from a DSP subsystem menu.
- ***SPA Formulation Decomposition***
  Each Kronecker product factor from the formulation is analyzed. Parameters required from the user establish the number of functional primitives to be used by the Scalable Operator and the internal organization of these constructs to address a specified computational structure.
- ***Computational Structure Composition***
  Using the Multiplication Operator and the required functional primitive constructs, computational structures are built and may also be stored in the FPL as functional primitives for larger structures.
- ***FPGA Implementation of Computational Structures using the Xilinx Implementation Tools.***
  Synthesis of the VHDL programs produce a *netlist* file as input to the Xilinx implementation environment which is then transformed into a bit stream to be downloaded into an FPGA device.
- ***Performance Assessment***
  Reports on area and speed values are indicators of performance and are a comparison tool between performance of the various structure implementations of an algorithm.

## 4. RESULTS

At the present time a HCS Generator environment is implemented (see Figure 3) composed of the following components:

**Kronecker–based SPA subsystems**
*DSP Subsystem*: Pre - determined Kronecker product representations of selected DSP algorithms; actually, the FFT algorithm [1].
*Custom System*: User generated algorithm by means of the FPL tool.

**Implementation Scheme**

User may select from different algorithm representations for a selected subsystem. Example: Cooley-Tukey, Pease, Stockham, Korn-Lambiote

**Parameter Section**
Required values for proper evaluation of Kronecker representation of selected DSP algorithm.

**Kronecker Formula Evaluation**
Display of the Kronecker expression based on previous parameters and scheme selection for a certain algorithm to be evaluated.

**Performance Report**
Information display on area and resource utilization on FPGA device as well as throughput and scalability extent.

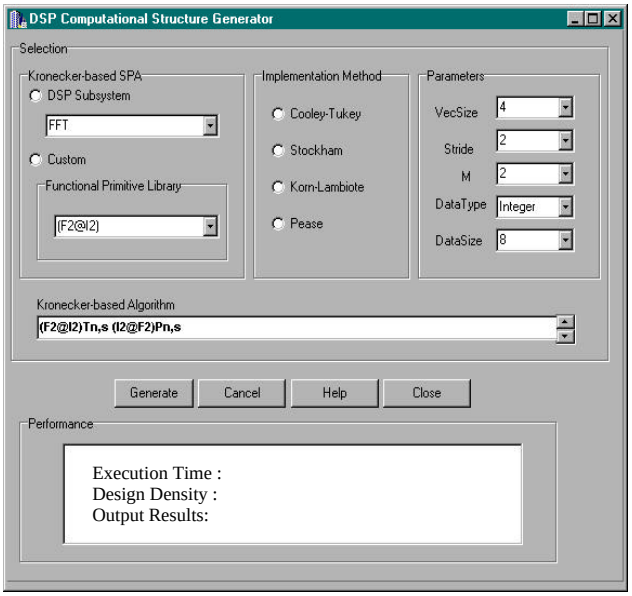


Figure 3: GUI for HCS Generator

## 5. FUTURE WORK

The first stage of our environment is designed for FPGA computational structure implementation of the FFT. Our future work consists on updating the functional primitives library (FPL) for general signal processing algorithms, i.e. Cyclic Convolution, FIR filters, etc., and suit the environment for general methods for these algorithm implementations. We will also include higher order functional primitives, known as composite primitives, to aid in the burden of small factor decomposition of large algorithms.

## 6. CONCLUSIONS

We have presented an environment with an identified methodology for the mapping of Kronecker–based signal processing algorithms to FPGA computational structures and have demonstrated that Kronecker products represent an alternative method for modular design of computational structures for signal processing algorithms. We have also shown that FPGAs reconfigurable properties and internal block structure provide an efficient framework for rapid prototyping of modular design.

## 7. REFERENCES


[1] D. L. Dai, S. K. S. Gupta, S. D. Kaushik, J. H. Lu, R. V. Singh, C.H. Huang, P. Sadayappan, R. W. Johnson "EXTENT: A Portable Programming Environment for Designing and Implementing High-Performance Block Recursive Algorithms",

[2] J. R. Johnson, R. W. Johnson, D. Rodríguez, R. Tolimieri, "A Methodology for Designing, Modifying, and Implementing Fourier Transform Algorithms on Various Architectures", IEEE *Transactions on Circuits, Systems, and Signal Processing*, Vol. 9, No. 4, 1990.

[3] L. Mintzer, " The FPGA as FFT Processor", *The International Conference on Signal Processing Application and Technology*, vol. 1, October 1995, pp. 657-661.

[4] G. Goslin, "Using Xilinx FPGAs to Design Custom Digital Signal Processing Devices", *Proceedings of DSPx*, Jan., 1995,

[5] D. Rodríguez, "Kronecker Product Algebra as a Tool for VLSI Implementation of the Discrete Fourier Transform", *Proceedings ICASSP 91*,

[6] B. Fawcett, R. Iwanczuk, "Techniques and Tools for FPGA-Based DSP Design", Xilinx Inc, August 1995, pp. 657-66