Signal Processing Implementations Using Simulink®

Alberto Quinchanegua, William Sánchez
Advisor: Domingo Rodríguez

Electrical and Computer Engineering Department
University of Puerto Rico, Mayagüez Campus
Mayagüez, Puerto Rico 00681-5000
albertoq@ece.uprm.edu, william.david@ece.uprm.edu

Abstract

This document presents a methodology for the implementation of signal processing applications based in a Kronecker product language to map a mathematical formulation in an algorithm formulation, and Simulink®, which is a dynamic system and simulation software based in MATLAB for modeling and high level simulation. Simulink® adds many features specific to dynamics systems while retaining all MATLAB general purpose functionality. A mathematical formulation can be expressed in terms of functional modules with defined input and output ports that can be interactively interconnected using Simulink in order to reduce the algorithm development and implementation time-line. Two important signal processing applications such as FIR filters and fast Fourier Transform are treated here as systems, that were implemented in hardware using a system generator toolbox, which translate a Simulink model in a hardware description language (HDL) for FPGA implementations.

1. Introduction

Simulink® is a dynamic system and simulation software that offers an interactive scientific and engineering environment for system modeling, analysis, and simulation. This environment is useful for the rapid implementation of a digital signal processing application in terms of functional blocks, and provides a high-level simulation. A functional block is a basic structure that can represent a function, or a specialized system, with defined input and output ports and customized parameters. In this work we concentrated in the implementation of digital signal processing algorithms using block diagrams for the data flow modeling, and algorithm testing and implementation. For this purpose, we introduce a methodology for the DSP algorithm development and hardware implementation, based in a Kronecker product language, in order to assist to a developer in the generation of hardware prototype systems. We define an algorithm development and implementation environment as an aggregated of the following tools: Simulink®, DSP microprocessors, and FPGA computing structures. We first introduce the mathematical formulations for two important applications in digital signal processing, such as FIR filters, and the fast Fourier transform; thus we proceed to map the mathematical formulations in algorithms expressed as interconnected functional blocks that can be translated in C code for DSP microprocessor implementations, or in VHDL code using a System Generator for FPGA implementations. Hardware implementation results of our methodology are presented using a Xilinx Virtex-II FPGA computing unit.

2. Modeling DSP Applications

Simulink® uses a set of libraries for signal processing to represent dynamic systems. The standard block library is organized into several subsystems, grouping blocks according with the behavior and it contributes with the design of new blocks by a developer, for increase the modularity of the applications. On the other hand, the reconfigurability of a developed model is not complicated because the "cut", "copy", and "paste" commands are available to include blocks by software. The **Figure 1** shows an example of a Simulink® model built with functional blocks.

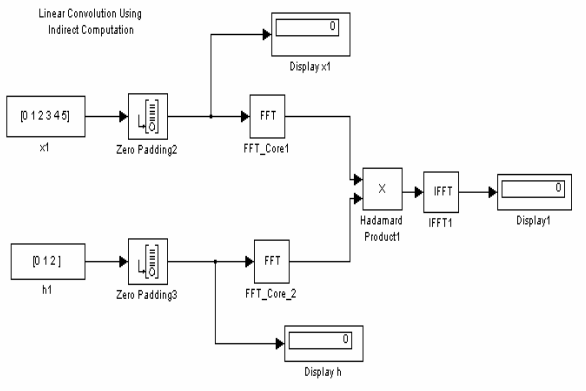

***Figure 1**. An example of a Simulink® model composed of interconnected functional blocks.*

In order to introduce the methodology for the model generation and the implementation of DSP applications we introduce the mathematical formulations for FIR filters and the FFT, using a Kronecker products language, and then we present the procedure to describe an algorithm in terms of functional blocks for testing, simulation, and code generation for hardware implementation.

2.1 Mathematical Formulations

2.2.1 FIR Filters

A simplest FIR system or operator, apart from the trivial system, i.e., the system represented by the identity operator I_N, is the system $T_h = T_{\delta_{\{1\}}}$, which turns into the shift operator S_N. This system is sometimes called the unit delay system because its digital electronics hardware implementation may be accomplished by using a single delay element, which acting on the unit sample signal $\delta_{\{1\}}$ results in $T_{\delta_{\{1\}}} = \sum_{j \in Z_N} \delta_{\{1\}}[j] S_N^j = S^1 = S_N$. The matrix representing the shift operator S_N is obtained by allowing the vector representation (with respect to the standard basis set $\left\{\delta_{\{k\}} : k \in Z_N\right\}$) of the signal $T_{\delta_{\{1\}}}\left\{\delta_{\{k\}}\right\}, k \in Z_N$, become the columns of the matrix. An important property of the S_N operator matrix is that any cyclic FIR system T_h may be represented by a matrix H_N which can be written as a sum of powers of the matrix S_N pre-multiplied by a diagonal matrix $D_{h[j]}$, as follow:

$$H_N = \sum_{j \in Z_N} D_{h[j]} S_N^j = \sum_{j \in Z_N} \left(h[j] \otimes S_N^j\right). \qquad (1)$$

2.2.2 Fast Fourier Transform

The Fourier matrix can be factorized in terms of sparse matrices represented by Kronecker products factors, which play an important role in the development and implementation of Fourier Transform algorithms. For example, the traditional Cooley-Tukey algorithm for the fast computation of the Fourier Transform can be represented in terms of Kronecker products factors as follows:

$$F_N = (F_R \otimes I_S) T_{N,S} (I_R \otimes F_S) P_{N,R} \qquad (2)$$

where N is the size of the FFT, $N=R.S$, and I_N is an identity matrix. In this case, N is a power of two. A new variant for the expression of the matrix F_N is derived using Kronecker product properties in order to compute a N-Point FFT, where N can be a non-power of two. The new mathematical formulation can be written as follows:

$$F_N = \left(U_2^T \otimes F_2 \otimes I_{2p}\right) T_{L,S} \left(I_{2p} \otimes U_2 \otimes F_2\right) P_{N,S} \qquad (3)$$

where $N = 4p$, and p is a prime. The expressions (2) and (3) allow us to identify functional primitives or basic structures that compose an algorithm. These functional primitives can be modeled as block diagrams, using the mathematical expression to define the data flow for the algorithm computation. For example, consider the Kronecker factor $\left(I_{2p} \otimes U_2 \otimes F_2\right)$ that represents a first stage in the FFT computation. Graphically we can determine the data flow generated by the action of the Kronecker factor over an input vector of length 4p, being p=1; then:

$$\left(I_2 \otimes U_2 \otimes F_2\right) \cdot x = \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes F_2\right) \cdot x \qquad (4)$$

$$\left(I_2 \otimes U_2 \otimes F_2\right) \cdot x = \begin{bmatrix} F_2 & 0 \\ F_2 & 0 \\ 0 & F_2 \\ 0 & F_2 \end{bmatrix} \cdot x \qquad (5)$$

where $F_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$.

The expressions (4) and (5) below represent the action of matrices F_2 over segments of the input vector x, where the operations over a same segment are duplicated, generating a data scattering stage. This scattering represents a parallel processing stage that permits to organize the stages of Twiddle factors present in the traditional Cooley-Tukey algorithm in a single parallel stage of Hadamard products. A diagonal matrix $T_{L,S}$ contains the twiddle factors. The same procedure explained below is carried out to identify the dataflow generated by the factor $\left(U_2^T \otimes F_2 \otimes I_{2p}\right)$, which is translated in a final stage of data gathering.

2.2. Functional Blocks

2.2.1 FIR Model:

In a real life a designer has to do the mapping from the mathematical representation of a FIR filter, to symbolic representation in Simulink®. From the mathematical formulation previously presented in the expression (1),

the designer can observe the implicit mathematical operations such as Add, Multiplication, and Delay and translate these in Simulink® functional blocks. For a 4-tap FIR filter implementation, Simulink offers the Delay Blocks, Multiplication Blocks, and Additions Blocks that the designer can interconnect to obtain the FIR filter structure for simulation and implementation as shows the **Figure 2**.

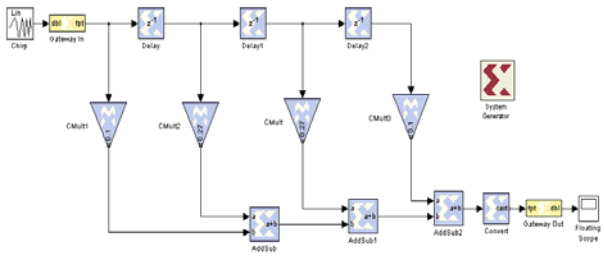

Figure 2. *Simulink model of a 4-tap FIR Filter.*

Notice in the figure below that the FIR filter structure has a repetitive structure conformed by a delay block, a multiplication block, and an adder block as shows the F**igure 3**. After the model simulation, the designer can make new functional blocks, creating new libraries with defined input and output ports, to be used in future applications.

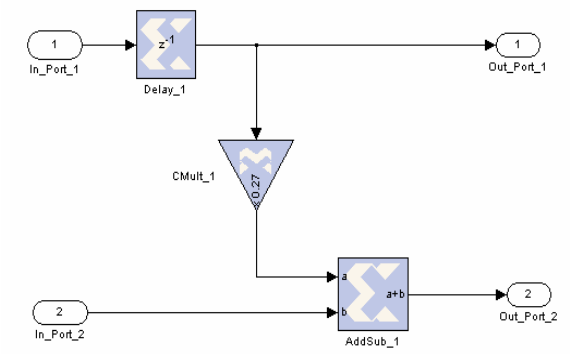

Figure 3. *Identification of a new functional block for the implementation of a FIR filter.*

2.2.2. 4*p*-FFT Model

The 4*p*-FFT algorithm can be represented in Simulink®, in terms of interconnected fuctional blocks for analysis and simulation. Each functional block is associated with a functional primitive, simulating the algorithm data flow in every stage of the FFT computation. The initial stage of the FFT can be implemented as a butterfly stage followed by a scattering stage as shows the **Figure 4**. A functional block representing the dataflow for the butterfly is shown in the **Figure 5.**

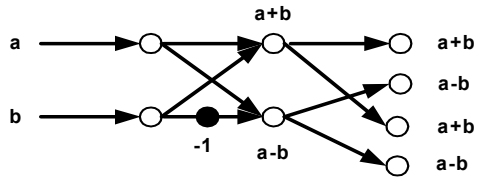

Figure 4. *Flow graph of a Butterfly followed by a scattering stage.*

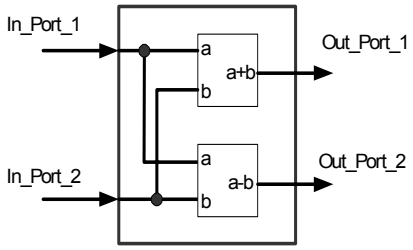

Figure 5. *Butterfly functional block for the implementation of the factor* $\left(I_{2p} \otimes U_2 \otimes F_2\right)$.

The others stages of the FFT computation are the Hadamard product or point-by-point multiplications, and the data gathering. Thus, the complete 4p-FFT algorithm can be modeled as shows the **Figure 6**.

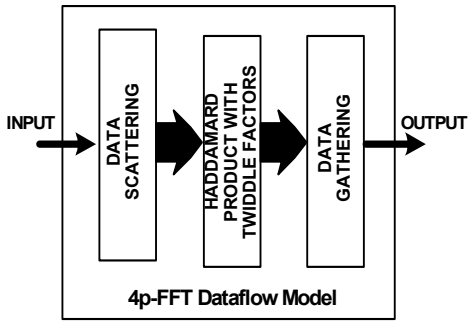

Figure 6. *Data flow model for the 4p FFT computation.*

3. Hardware Implementations

This section presents preliminary results of the FPGA hardware implementation of a 4-tap FIR filter, and a new variant of the traditional Cooley-Tukey FFT algorithm for lengths that are not necessarily power of two. The FIR filter previously presented in the **Figure 2** was implemented in a Virtex-II FPGA using a System Generator toolbox for the generation of VHDL code. The implementation results of the FIR filter are compared with a previous implementation of the same 4-tap FIR filter in a TI-C6711 DSP microprocessor and summarized in the **Table 1**.

Table 1. *Hardware implementation results of a 4-Tap FIR filter.*

ITEM	***Virtex-II FPGA***	***TI-C6711 DSP µ-Processor.***
Data Bus Input	1~n* Bits Fixed Point	32 Bits Floating Point.
Data Bus Output	1~n* Bits Fixed Point	32 Bits Floating Point.
System Clock Period	10 nanoseconds (max. 100 MHz)	6.7 nanoseconds (150 MHz)
Latency	1 clock cycle**	49 clock cycles
Simulation	Logic Devices, Bit stream.	Registers, Bytes, Words.
Programming Language	VHDL	C, assembly
Scalability	Data Bus Width	Fixed architecture.

** n≤ 32 using Xilinx Logic Cores.*
*** Initial latency: 11 clock cycles.*

The 4*p*-FFT algorithm is based in a Kronecker formulation that permits to organize the stages of Twiddle factors present in the traditional Cooley-Tukey algorithm in a single parallel stage of Hadamard products, reducing the latency time generated by every multiplication stage. The algorithm was simulated in Simulink® and the model translated in a hardware description language (VHDL) using a System Generator Core for FPGA implementation. The **Figure 7** shows a block diagram of the developed hardware structure composed of interconnected functional blocks.

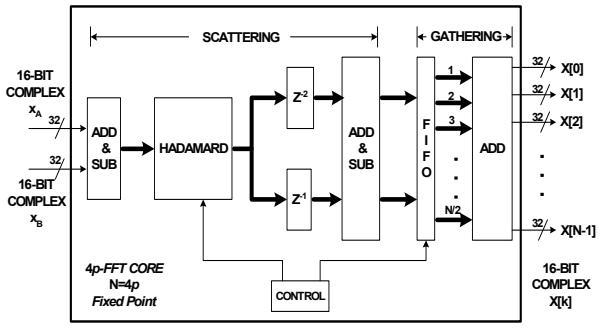

Figure 7 *Block diagram of the 4p- FFT hardware structure.*

The 4p-FFT model was implemented in hardware using a Xilinx Virtex-II FPGA and the results are presented in the **Table 2**.

Table 2. *FPGA implementation results of the 4p-FFT model.*

N-Complex Point	**Slice Consumption**	**Initial Latency Time (Cycles)**	**System Clock Period**
8	1198	16	10.287ns
12	2516	19	10.287ns
16	4165	22	10.287ns

4. Conclusion

We presented a methodology for the formulation of computing methods for digital signal processing, based in a Kronecker products decomposition technique that allow us to map a computing method in the form of algorithm formulations, and to find new variants to adapt the algorithm to specific hardware computing architecture. We represented the algorithms in terms of functional blocks for modeling and simulation using Simulink®. The modeled DSP applications were implemented in a FPGA using a system generator that translated a Simulink model in a Hardware description language for FPGA implementation. Also, a Developer's Kit for Texas Instruments toolbox can be used to generate machine code for a TI-C6000 DSP Processor from a Simulink® model.

References

[Johnson90] 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.

[Rodriguez02] D. Rodríguez, A. Quinchanegua, H. Nava. "Signal Operator Cores for SAR Real Time Processing Hardware," IEEE IGARSS 2002, Toronto, Canada, July 2002.

[Rodriguez91] Domingo Rodriguez, " A new FFT algorithm and its implementation on the DSP96002," ICASSP '91, Vol. 3, pp. 2189-2192, May 1991.

[Rodriguez91] Domingo Rodriguez, " Tensor product algebra as a tool for VLSI implementation of the discrete Fourier transform," ICASSP '91, Vol. 2, pp.1025-1028:May 1991.

[Reyneri01] L.M. Reyneri, F. Cuccinotta, A. Serra, L. Lavagno. "A Hardware/Software Co-design Flow and IP Library Based of Simulink™," Design Automation Conference, pp. 2101-2106, Vol. 4. 2001.

[Xilinx03] Xilinx, Inc, http://www.xilinx.com

[Math03] The Mathworks, Inc, http://www.mathworks.com

[TI03] Texas Instruments Inc, http://dspvillage.ti.com