# From Matlab/Simulink Models to Prototype Implementation: A Communication Systems Development Environment

*M. Varsamou, P. Savvopoulos, N. Papandreou*
*and Th. Antonakopoulos*

The Nordic MATLAB Conference 2003 – NMC 2003

COPENHAGEN, DENMARK, OCTOBER 2003

# From Matlab/Simulink Models to Prototype Implementation: A Communications Systems Development Environment

Maria Varsamou, Panayiotis Savvopoulos, Nikolaos Papandreou and Theodore Antonakopoulos
Academic Research Computer Technology Institute - CTI
61 Riga Feraiou Str., 26100 Patras, Greece
Department of Electrical Engineering and Computers Technology
University of Patras, 26500 Rio - Patras, Greece
e-mail: {varsamou, psavvop, npapandr, theodore}@loe.ee.upatras.gr

## Abstract

*The aim of this paper is to present a flexible and versatile environment for developing, analyzing and prototyping data communication and signal processing systems. This environment is based on the Matlab/Simulink tools and a reconfigurable hardware platform that includes reprogrammable and digital processing circuits. The hardware platform communicates with the Matlab workspace via a dynamic data exchange and synchronization mechanism, which enables the interaction between the model running on the Matlab/Simulink and the hardware/software modules developed by the user. This development environment provides a flexible test-bench that facilitates the gradual implementation of a high-level system model to an actual prototype.*

## 1. Introduction

Prototyping of communications systems is a demanding task that involves several discrete design steps. Initially, an analytical model of the system has to be developed, the various algorithms have to be designed and the system behavior needs to be verified. As a next step, the respective prototype which combines a number of hardware and software functional modules has to be implemented and tested in terms of its consistency to the specifications of the analytical model. Due to the complexity of their algorithms, modern communications systems need long development and testing time for the completion of the prototype. Regarding the modeling and testing of a communications system, Matlab/Simulink tools offer a high performance simulation environment that supports the development and analysis of complex multi-domain models.

In this paper, we discuss an environment that provides an effective design and test approach, by exploiting the high-performance simulation and modeling capabilities of the Matlab/Simulink tools and the flexible modular hardware architecture of a prototype platform. The design methodology involves the use of the Matlab/Simulink tools for building and verifying the analytical model and then mapping selected system blocks into hardware and/or software modules on the prototype platform. These blocks are being replaced by special library functions that are responsible for the communication and synchronization with their circuit counterparts. Throughout the development process we can take advantage of the same Matlab-based testing tools in order to verify the proper behavior of the various integration steps. This procedure continues until the high-level model functions have been integrated into a complete prototype.

The rest of this paper proceeds as follows: In Section 2 we describe the prototype design approach. In Section 3 the hardware platform and its interconnection with the Matlab/Simulink tools is presented, while in Section 4 specific synchronization issues between the simulation tools and the digital processors are discussed. Finally, in Section 5 an application example that highlights the functionality of the proposed environment is presented.

## 2. The Prototype Design Methodology

The prototype design methodology, as it is shown in Figure 1, is based on a top-down approach that defines how a high-level system model, developed using the Matlab/Simulink tools, can be transformed to a prototype.

During the initial phase of the proposed design approach, the system specifications are defined and all necessary information about the system architecture, its functionality and complexity is collected. Based on this information, new algorithms are developed and a high-level Matlab/Simulink model is built, composed by standard library blocks and custom functions [1]. The model's functionality is validated

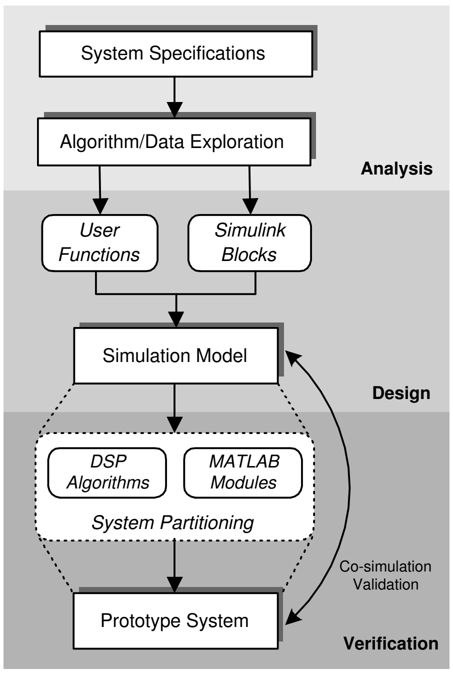

**Figure 1. The prototype design methodology.**

and optimized through simulation and a first estimation of the implementation complexity is possible.

After the model design has been completed, various submodules are progressively mapped into hardware/software components on the hardware prototype platform. Communication between the high-level simulation model and the hardware platform is accomplished through data exchange and synchronization mechanisms. The mixed-type system is verified using the same testing tools and procedures that have been used to validate the high-level model. This approach allows the user to perform low level debugging, while simulation results are being collected and compared with the respective results of the high-level model. Optimization is performed for every submodule of the communication system individually and at the end, a fully functional system that is consistent with the initial specifications has been developed.

## 3. The Hardware Platform Architecture

In order to implement a low-cost and flexible prototype environment, we developed a hardware platform that is based on reprogrammable logic and a DSP processor. Intercommunication between the hardware platform and a computer, that hosts the Matlab/Simulink tools, is performed via the PCMCIA interface. A complete development environment consisting of two PCs with two hardware platforms is demonstrated in Figure 2. This structure corresponds to a pair of communicating devices and provides the capability of designing flexible component-level architectures, thus enabling the implementation and verification of several end-to-end communication applications.

Each hardware platform is based on the high-performance floating point TMS320C6711 [2] processor of Texas Instruments with 900 MFLOPS processing power, an analog front-end that includes two ADC and two DAC channels, and an FPGA module with 8 kwords of external Dual Port RAM. The FPGA's internal logic implements the PCMCIA interface and extends the available memory space that can be accessed by the Matlab workspace via an appropriate I/O device driver. The data exchange and synchronization between the model and its blocks that are mapped into the prototype platform, is achieved through Matlab/Simulink custom functions that associate workspace variables with memory locations and structures at the interface memory space. The DPRAM contains all necessary user and control data required for the efficient implementation of the mixed-type model. In the current version of the prototype platform, all subsystems are implemented as DSP functions that are synchronized by the Matlab environment which is the master throughout these transactions. Figure 3 shows the actual hardware prototype set-up used in this work.

For more demanding applications, the presented prototype platform can be easily extended in order to include additional hardware modules that may be accessed by the Matlab/Simulink functions using the same memory space. This is achieved by using internally some dedicated I/O ports. Such an extension enhances the environment's flexibility and enables the exploration of more elaborate designs, as resource demanding parts of a communication or signal processing model may be substituted by efficient hardware implementations.

## 4. Synchronization Issues

In a mixed-level design that includes a Matlab/Simulink model, whose certain parts are implemented as DSP functions in the hardware platform, synchronization issues arise that play a key role in the effective and proper prototyping approach. Synchronization has two different aspects: synchronization of the several functions performed at the same DSP processor, and synchronization that is related to the communication and data exchange between the Matlab workspace and the DSP environment.

### 4.1 Multi-threading in the DSP environment

The progressive substitution of several Simulink blocks by their respective DSP implementations, results in the de-

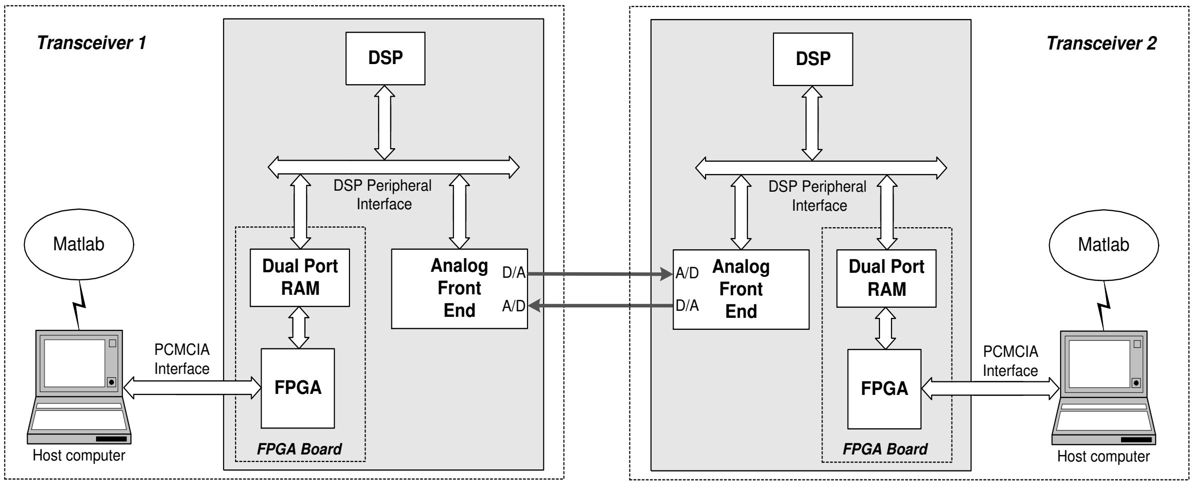

**Figure 2. The architecture of the prototype environment.**

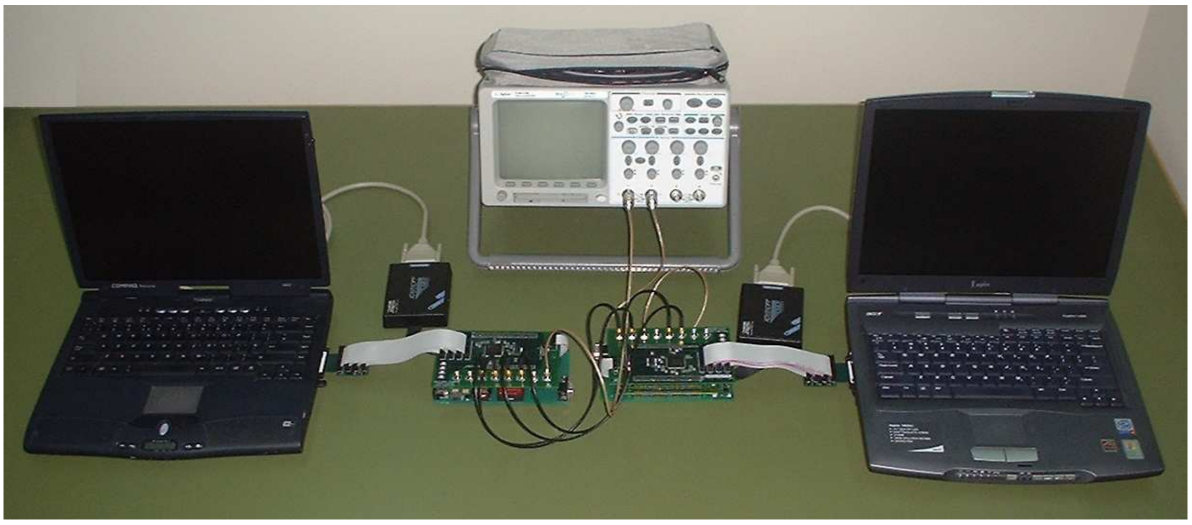

**Figure 3. The hardware prototype set-up.**

velopment of DSP functions that perform a number of different functions simultaneously, often in response to external events, such as the availability of data or the presence of a control signal. These functions are called threads and multi-threaded programs run on a single processor by allowing higher-priority threads to preempt lower-priority ones and by allowing various types of interaction between threads, including blocking, communication, and synchronization. The used DSP processor supports multi-threading applications through DSP/BIOS that is supplied along with the Code Composer Development Suite [3]. DSP/BIOS is a scalable real-time kernel, which is designed for applications that require real-time scheduling and synchronization, host-to-target communication, or real-time instrumentation. It provides preemptive multi-threading and system-level services such as memory management, communication mechanisms and interrupt handling. Using these kernel features, distinct submodules of a Matlab/Simulink model can be mapped to independent or cooperative DSP functions that are executed concurrently.

## 4.2 Synchronization between the Matlab/Simulink model and the hardware platform

Synchronization should be achieved not only among distinct DSP functions, but also between the Matlab/Simulink functions and the DSP modules, in order to ensure that the data-flow through the model's different stages is consistent with the system's specifications.

The complexity of the synchronization problem varies depending on the real-time data exchange requirements between the model and the hardware platform. The simplest case is when a certain block of a Matlab/Simulink model, like an FIR filter, is replaced by a respective DSP function. Then, synchronization is achieved by a hardware control mechanism, or by control information exchanged using the Dual Port RAM. Matlab produces the data to be transferred to the DSP module, and then seizes its execution until the response is received. A different scenario is to use Matlab/Simulink for monitoring the behavior of a communication or signal processing system implemented on the hardware platform, via high complexity blocks, such as FFT-Scopes or Averaging-Spectrum-Analyzers. In this case, blocks of information should be collected periodically by the Matlab model so that they can be processed and presented. To make that feasible, a more sophisticated procedure is defined, involving observation of control parameters and buffer manipulation by both Matlab and DSP.

The most challenging synchronization issues arise when real-time data exchange between Simulink and DSP is needed. For example, when a communication channel with certain characteristics is emulated and a Simulink application includes modulators/demodulators and/or encoders/decoders. It is obvious that in this case, there must be a continuous data-flow from Matlab to the hardware platform and vice versa, so that realistic emulation takes place. The difficulty in this case stems from the fact that although the hardware platform constitutes a hard real-time system, the simulation times in Simulink cannot be explicitly determined since they depend on the complexity of the model, on the computer's processing power and the data exchange rate achieved at the PCMCIA interface. As a consequence, there is an upper bound to the maximum data transfer rate that can be achieved between the high-level model and the hardware platform.

On the other hand, when the burden of the calculations lies on the DSP, leaving just a light model in Simulink, which therefore might be too fast, an adaptation to the predefined rate is feasible via a custom block adding variable delay to the simulation loop. Continuous data exchange is accomplished through an elastic circular buffer, which is placed in the Dual Port RAM and is shared by both Matlab and DSP, as it is shown in Figure 4. Due to the real-time nature of the data transmission on a communication channel, the buffer should never be overflown either by Simulink or by DSP, since in this case, inaccurate simulation results will be obtained. This requirement along with the fact that the buffer is accessed by two applications on the same time necessitates the existence of a well-defined buffer manipulation process. Apart from the buffer itself, there are two pointers in the DPRAM, one that is updated by Matlab and addresses the last location that has been accessed by the Matlab and another that is updated by the DSP and addresses the last location that has been accessed by the DSP. A custom Simulink block was developed, which ensures that the buffer never encounters underflows or overflows. During simulation, the Simulink block observes the difference between these two pointers, thus determining the optimum rate at which the buffer has to be accessed. Depending on the size of this buffer, the block adapts the Simulink's processing time accordingly.

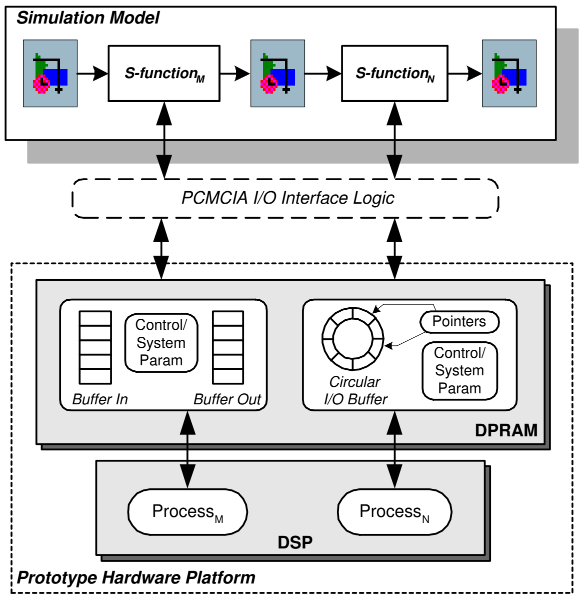

**Figure 4. Runtime communication between Simulink and DSP modules.**

## 5. Application Example

In this section we present an example that demonstrates the applicability of the presented design methodology and hardware prototype platform in the development and implementation of a digital communications system. Figure 5 shows a general block diagram of a QAM (Quadrature Amplitude Modulation) [4] communication system that consists of the transmitter, the far-end receiver and an AWGN channel emulator, used to add Gaussian noise to the transmitted signal. The transmitter consists of the binary random source generator, the QAM constellation encoder and modulator and the Tx-filter used to shape the output signal. The reverse functions are executed at the receiver, in order to demodulate the incoming signal and extract the binary data information.

Following the design methodology presented in Figure 1, we have decomposed the system under development into functions that are executed in the Matlab/Simulink environ-

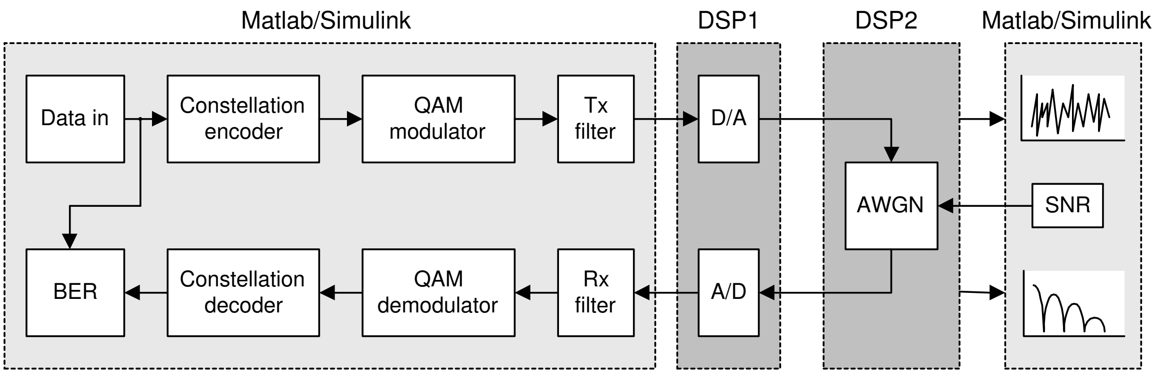

**Figure 5. The functions of the QAM Modulator-Demodulator experimental set-up.**

ment and functions that are executed in the DSP device of the prototype platform. In particular, the basic blocks of the transmitter and receiver are developed in Matlab/Simulink functions and a custom library is used to transfer the output and input samples to/from the DSP device, where the D/A and A/D conversions are performed. A second DSP platform is used to add AWGN to the transmitted signal. The noise level is determined "on-the-fly" by the user via the Matlab/Simulink interface with the DSP platform. In this second Matlab/Simulink platform, some testing modules are used for measuring the quality of the input and output channel signals. The complete experimental set-up is the one presented in Figure 3. Using the data and synchronization interface between the Matlab/Simulink environment and the prototype platform we can exploit the simulation tool's build-in functions (e.g. signal and spectrum scopes, scattering plots, BER tester) in order to provide real-time visualization of the communications system performance.

The mixed-level example of Figure 5, demonstrates the development of a full communications system, consisting of Matlab/Simulink models and scope functions along with hardware prototype circuits. Moreover specific modules of the QAM transmitter and receiver can be further replaced by hardware/software counterparts, in order to develop a complete QAM transmitter/receiver in the DSP platform.

## 6. Conclusions

In this work we presented a versatile development environment for prototyping a communications system starting from its Matlab/Simulink model and exploiting a flexible hardware platform. The presented methodology enables the designer to start the development of a system from building up a high-level simulation model, perform extensive test and verification and continue the road-map to the final prototype by substituting various system submodules with their respective hardware/software implementations on the prototype platform. The key of this approach is that the same testing procedures used for verifying the model can be reused for testing the proper functionality of the prototype system. Furthermore, this environment can be used for educational purposes on system modeling, verification and implementation and for demonstrating the basic aspects of the communication theory as well.

## References

[1] The Mathworks Inc., Writing S-Functions, Revised for Simulink 5.0 (Release 13), July 2002.

[2] Orsys GmbH., User's Guide Micro-Line C6122CPU/ C6711CPU/ C6712CPU, High Performance Digital Signal Processor Family, Rev 4.02, Mar. 2003.

[3] Texas Instruments, Inc., TMS320 DSP/BIOS User's Guide, Nov. 2001.

[4] F. Xiong, *Digital Modulation Techniques*. Norwood: Artech House, 2000.