Simulink Block Library for Fast Prototyping of Reconfigurable DSP systems

Tamás KOVÁCSHÁZY, Gábor SAMU, Gábor PÉCELI
Budapest University of Technology and Economics,
Department of Measurement and Information Systems, Budapest, Hungary
Email: {khazy,samu,peceli}@mit.bme.hu

Abstract *– This paper presents a block library for Matlab/Simulink that allows fast prototyping of reconfigurable DSP systems. Up till now no similar software package was available. The block library supports the construction of reconfigurable discrete time linear and non-linear systems from reconfigurable digital filters using various filter structures, state-space form implementations, polynomial filters, and PID controllers. The paper lists the requirements for the block library and introduces the main implementation related decisions that allows the block library to meet these requirements. An example illustrates the usage of the block library.*

Keywords *– reconfigurable DSP systems, transient management and reduction, Simulink block library*

I. INTRODUCTION

Reconfigurable digital signal processing (DSP) systems are in the center of research interest [1,2,3] because they can efficiently model and control time-variant and/or non-linear complex, distributed systems, e.g., industrial processes, vehicles, etc. The design and implementation of such reconfigurable DSP systems require multi-domain knowledge, i.e., software and hardware engineering, digital signal processing, fault-tolerant systems, networking, etc. To support this complex design task, it is necessary to have adequate software support, which should hide the peculiarities of all the domains except one, for the non-specialists and allow these experts to work together.

Matlab/Simulink is one of the most widely used tools for DSP system simulation and design. Simulink makes possible to capture complex system using a hierarchical signal flow graph (SFG) notation, where atomic and compound signal processing blocks are connected together to form compound blocks, and finally the whole system under simulation or design. Unfortunately, as nearly all other comparable tools, Simulink lacks support for reconfigurable systems that are composable at run-time. Although, it is possible to build such a system using the currently available blocks but the resulting component cannot be used as compound component to build more complex systems in an efficient way. In addition, run-time block substitution is not supported at all. Therefore, it is necessary to build a Simulink block library for reconfigurable systems. The developed block library can be used not only as a fast prototyping tool, i.e., which allows to check the various alternative system implementations using simulations, but it can be considered also as a non-real-time design and implementation prototype for hard real-time reconfigurable DSP systems. The block library is in its final development stage, and after the final testing, it is planned to be released as free, downloadable software module for Matlab/Simulink.

In Section II of the paper a reconfigurable system architecture is introduced, in which our block library implements certain functionalities. Section III introduces the block library, some requirements, and some major internal architectural decisions. An illustrative example is shown in Section IV. Section V details some possible future development directions. Finally, conclusion are drawn in Section VI.

II. RECONFIGURABLE SYSTEM ARCHITECTURE

The inherent complexity of reconfigurable systems can be dealt with by introducing a component-based, layered architecture. One possible partitioning is shown on Figure 1, which is used in the FACT framework [4]. In this paper, the right side of Figure 1 is in the center of interest, because all the transient management and reconfiguration related components/layers are there. The components in Figure 1 are the following:

- Global supervisory controller (GSC),
- Reconfiguration manager (RM),
- Local Blocks (LB),
- Sensors and actuators,
- System identification,
- Data plane.

This paper introduces a Matlab/Simulink block library providing blocks primarily applicable on the LB layer, but they may be used also on the RM layer. The reconfigurable blocks can be used as the model for physical systems, sensor,

and actuators, which should be changed during simulation in most practical cases also. The block library does not intend to solve the whole problems of reconfigurable system design, it concentrate on the reconfigured systems and transient management exclusively. However, it is necessary to introduce the tasks of various blocks on the right side of Figure 1 and their relationship to the process of reconfiguration.

The GSC component does not directly deal with transient management and reconfiguration, because its main role:

- To capture global modes of the system,
- To produce global, high level control signals and events (primarily used by the RM and LB components, and not by the physical system).

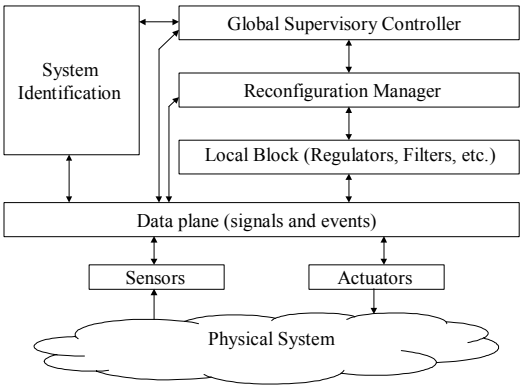

Figure 1. Components of the reconfigurable system

The RM component:

- Acts as an intermediary between the GSC and LB:
  - By mapping GSC modes to LB modes (not necessarily a one to one mapping),
- Does global transient management,
  - By incorporating transient management related information and temporary modes, and synchronizing the transient management related activities of LBs,
  - By doing transient related decision-making and design.

The LB components encapsulate the systems to be reconfigured, e.g., filters, regulators, etc., and all other local block specific functionalities, such as design from low-level implementation specific parameters to more complex, typically non real-time, design from specification. In addition, the main functionality of LBs is to provide low-level transient management, and safe block reconfiguration. Therefore, the LB components:

- Are connected to signals and events on the data plane,
- Operate on and produce signal and event inputs and outputs based on local criterion by incorporating filters, regulators (the global criterion are incorporated by the configuration),
- Consist of the local logic (local supervisor) to do local (and implementation specific) reconfiguration and local transient management,
- Incorporate design procedures.

In distributed systems requiring distributed digital signal processing, the LBs may be real controller boxes near the plant, connected to certain sub-systems of the physical system; for example, one or more LBs may be assigned to a control surface of an airplane.

The data plane acts as a virtual, real-time, configurable "wiring closet" to distribute signals and events to the interested components coming from the signal sources. This data plane can be realized by a low-level communication network with such higher layer protocols on each node (implementing the LB, RM, GSC, etc.), which provide guaranteed real-time performance [5][6].

III. REQUIREMENTS AND IMPLEMENTATION

The development of the block library was primarily driven by the special requirements of reconfigurable systems; therefore, it is necessary to clearly state all the main requirements and introduce the solutions, which assures that the requirements are met.

A. Target Systems

The implemented block library supports a limited set of target systems, i.e., systems to be reconfigured, in its present form. The current list of supported target systems is the following:

- Reconfigurable linear digital FIR and IIR filters using the direct form, parallel form, and resonator-based implementations [7],
- Reconfigurable state-space form linear systems,
- Reconfigurable PID controllers,
- Polynomial filters [8].

These target systems are the most common reconfigurable system types used in the literature and; therefore, they are essential components for basic experimentations. The block library can be extended by new target systems. The new target systems must follow the standards of the block library, i.e., they need to be implement using a common interface.

All of these target systems implement a quite generic system interface to Simulink using standard Simulink data types and a custom, block library specific objects explained later in the Section III.D. The configuration and initial state related information can be specified in a static, non-reconfigurable way as the property of the block also. The interface needs to provide access to the following inputs and outputs:

- System inputs and outputs (standard Matlab/Simulink scalars, vectors, or matrices),

- System initial internal state input (vector),
- Initial internal state load enabler input (scalar),
- Internal state output (vector),
- System configuration input (custom object),
- System configuration load enabler input (scalar)

Figure 2 shows a resonator block as example, with one input (*u*), two outputs (*yfbs*, *ys*), initial internal state input and its load enabler input (*xi* for the initial internal states, *xv* enabler input), internal state output (*xo*), and finally, system configuration and configuration load enabler inputs (*RP* for the resonator coefficients, *pw* as enabler input).

The target system blocks implement interfaces to specify their configuration and internal states run-time to allow run-time reconfiguration. All these operations can be performed only in an atomic way; therefore, the blocks, their internal states, and their configuration are always kept in a consistent state [4]. New configuration or initial internal states are only loaded when their corresponding load enabler input is signaled. The enabler inputs in the block library are programmable, i.e., the user may select various trigger conditions to make the block load the supplied configuration or initial internal state, in addition, target specific internal reconfiguration and transient management methods can be also executed. The default trigger condition is "above 0", but "below 0", "rising edge", "falling edge" or "any edge" are also implemented to accommodate various types of signal sources. Most cases these enabler signals are generated by a state machines [4] implemented as described in Section III.E.

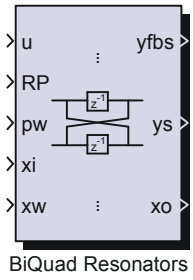

Figure 2. Resonator block used in resonator-based filters as an example block to show the standard interfaces of target system blocks

The target system blocks are supplemented with some representative design procedures [4] that makes possible to compute the coefficients of these target systems from other system specifications. For example, design procedures to compute resonator-based filter coefficients from transfer functions are available. The design procedures have also a quite common interface; they have the following inputs and outputs:

- System configuration output (custom object),
- System specification input (custom object and/or standard data type),
- Design procedure enabler input (scalar)

These design procedure functions have enabler inputs also, because most cases the design activity has high computational complexity, and they are used rarely, only before and sometimes during reconfiguration. However, the computational model of Matlab/Simulink requires the execution of all blocks in all iterations, which would result in the computation of these design procedure blocks in all iterations making efficient operation impossible. The design procedure; therefore, are programmed to do their functionality only when the enabler signal triggers them to do so, otherwise they return immediately. Their outputs are kept as they were after the last computation, in other words, they store the last computed configuration as an internal state.

B. Transient Management

Reconfiguration transients are identified as one of the major problem to be solved in the context of reconfigurable systems [1,9,10]. Therefore, transient management methods are developed to reduce the transients to an acceptable level.

The most widely known methods of transient management are [10]:

- One step reconfiguration,
- Multiple step reconfiguration with the gradual variation of the intermediate configurations using interpolation (series of one step reconfigurations),
- Input cross-fading methods,
- Output cross-fading methods,
- State variable update methods,
- Anti-transient signal injection.

The block library implements all of these transient management methods in a target system specific way for the appropriate target systems because all target systems have unique aspects [4], which prohibit the design and implementation of generic reconfiguration methods. Unfortunately, for the same reasons, it is not possible to formalize even a generic interface for the transient methods contrary to the target systems.

The implemented methods should support reconfiguration control and synchronization by letting higher-layer components to control the process of the reconfiguration [4]; in other words, they may have also reconfiguration control/enabler inputs, which specify certain aspects of their behavior during reconfiguration. These inputs can be also used to minimize computational resource utilization as it is done for the design procedure; specifically, to execute the transient management blocks only when it has to be executed, i.e., during reconfiguration. In addition, some transient management methods supply their own control (enabler) signals for the design procedures and the target systems.

All implemented reconfiguration methods are supplemented by an illustrative example in the block library, which act as design patterns helping the user to start experimentation with new ideas.

C. Efficient implementation for fast prototyping

Reconfigurable DSP systems require more resources compared to classic DSP systems because there are complex interactions between components and these components require resource intensive algorithms. In addition, large number of experiments may be required during the design phase due to the fact that research on reconfigurable system is in an early phase, and there are very few well-established design algorithms. However, this early phase of research also ask for the fast inclusion of new algorithms into the block library call for exceptional extendibility properties.

Our primary objective was to implement existing reconfigurable algorithms in an efficient way, and experiment with various open system parameters; therefore, all blocks are implemented in C++ using the MEX programming interface of Matlab. This solution has also some other advantages, such as:

- It makes porting the C++ code to real-time platforms very easy, because the kernel of computations is written in a computer language supported by the common real-time developments environments; therefore, only the interfaces of these kernels needs to be rewritten.
- It allows the objects to interface to other software packages than Matlab/Simulink or to distributed systems.

However, this solution makes inclusion of new algorithms into the block library more complex than creating homogenous Matlab/Simulink blocks. To ease the development of new blocks a detailed extension guide and examples are provided.

The block library is developed under Windows using Matlab 6.5; therefore, currently is available under these platforms.

D. Object transport on the SFG level

Reconfigurable system components communicate by not only using time series of scalars and vectors as classic digital signal processing systems, but complex compound objects are sent to other components, such as configuration of components, or system specification, which cannot and/or should not be mapped to scalars and vectors, but must be represented by special typed objects [4]. These objects needs to implemented in a way, which allows the connection of publisher block's output to the inputs of the subscriber blocks using standard Simulink connection objects, in essence, the transport of objects directed by the SFG specified in Simulink.

Matlab/Simulink supports construction of objects, but it is found to be inefficient from the point of view of performance in the Simulink environment, and it does not provide the expected flexibility and hard to use in Simulink. Therefore, in our implementations object are implemented in C++ and only references (pointers in our case) are passed based on the SFG specified in Simulink. The applied method is transparent, and compatible with other blocks in Simulink, and it also needs very limited resources. Universal support blocks provide access to the member variables of these objects. By using these block, it is possible to select member variables or construct object from Matlab data types like scalars and matrices.

Objects are passed by value in classic real-time distributed systems typically, i.e., they are copied from block to block using the communication infrastructure provided by the distributed systems. From this point of view, our decision to pass objects by reference seems to be inappropriate at first, but here we must note two of the previously mentioned premises of our design decisions again:

- Our block library is designed for efficient Matlab/Simulink fast prototyping on a single machine primarily, not for the seamless realization of real-time distributed systems,
- The blocks are not intended to be plugged directly to distributed systems, but their core algorithm implementations may be easily ported to other frameworks, e.g. Real-Time CORBA and/or OCP [5], by rewriting their interfaces according to the requirements of that framework.

E. Mixed time and event-driven operation

In typical applications, the time-driven reconfigurable DSP components are reconfigured by event-driven systems interpreting stimulus coming from other higher-layer components such as on-line identification and fault-detection and isolation, and from user input. Event-driven system components can be described using Stateflow in Matlab, using state-chart formalism of Harel [11].

The components of the block library are implemented to be compatible with this mixed mode of operation, i.e., they can be disabled and their operation can be controlled by signals supplied by Stateflow blocks. Since the computational model of Simulink executes all blocks in all solver iterations, it is absolutely necessary to let these components to be disabled, because some resource hungry operations need to be executed only during reconfigurations.

IV. ILLUSTRATIVE EXAMPLE

Figure 3 shows an illustrative application example of the block library. The example was set up to demonstrate the effectiveness of the anti-transient signal injection transient management scheme [10].

Therefore, two identical state-space type target systems are placed on the model, reconfigured the same way using the one-step reconfiguration [10], i.e., changing the coefficients of the system without modification of the internal states. An "Anti-Transient Signal Injector" (ATSI) block is placed and interconnected with one of the blocks, to reduce the transients caused by this block. The ATSI block gets the configuration

of the target systems, and some constants that define the span of the anti transient signal injection in the time domain. The system is started up, operated, reconfigured and stopped by a "Reconfiguration Manager" Stateflow block. The outputs of the blocks are plotted on a Simulink plotter.

The systems before and after reconfiguration are specified by the "Filter selector" from the user interface of Simulink by dialog boxes. The specifications are transformed to the appropriate state-space coefficients, i.e., $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ matrixes by a "Filter Designer" block, which is parameterized to do this transformation by dialog blocks. The old system is a 4th order low-pass Butterworh type IIR filter with the cut-off frequency of $0.2\,f_s$. The new system is a 4th order low-pass Chebyshev tye IIR filter with the cut-off frequency $0.1\,f_s$, and pass-band ripple of $0.2\,dB$. In other words, the system is reconfigured to a more stringent specification.

The resulting system outputs can be examined on Figure 4. The figure shows that by applying anti-transient signal injection it is possible to reduce the reconfiguration transients caused by the one step reconfiguration, if the detailed state-space models of the target system are known.

V. FUTURE DEVELOPMENT DIRECTIONS

The block set supports a limited set of target systems, design procedures, and transient management blocks. As the research on reconfigurable systems develops new blocks are to be added to the block library. In addition, after the planned release of the block library, the feedback and development efforts of other user will shape the block library considerably.

Support for Real-Time Workshop is under investigation, which may be implemented in the next version of the block library. By providing Real-Time Workshop support, the users of the blocks library would be able to generate real-time code for PC hardware, DSPs, microcontrollers, and real-time operating systems (RTOS) from their Matlab/Simulink model directly.

The block library is currently operates under Windows and Matlab 6.5, porting it to other operating systems is not planned but support of newer version of Matlab will be provided.

VI. CONCLUSIONS

A block library is presented for Matlab/Simulink that allows fast prototyping of reconfigurable DSP systems. Up till now no similar software package was available. The block library supports the construction of reconfigurable discrete time systems from reconfigurable digital filters using various filter structures, state-space form implementations, and PID controllers. System design and transient management blocks make the reconfigurable block library complete.

The block library can be also used as a framework and design pattern for other researchers implementing other type of reconfigurable systems under Matlab/Simulink, or by researchers designing reconfigurable systems in other environments.

The block library is implemented using C++ and Matlab's MEX programming interface on the Windows platform because Matlab/Simulink does not support the construction of reconfigurable systems in an efficient way.

ACKNOWLEDGEMENTS

The research described here was sponsored, in part, by the Defense Advanced Research Projects Agency (DARPA) (US) under agreement number F33615-99-C-3611 and by the Hungarian Office of Higher Education Support Programs under contract FKFP 0654/2000.

REFERENCES

[1]] Janos Sztipanovits, D. Michell Wilkes, Gabor Karsai, Csaba Biegl, and Lester E. Lynd, "The Multigraph and structural adaptivity," *IEEE Transactions on Signal Processing*, vol. 41, no. 8, pp. 2695–2716, August 1993.

[2] Gábor Péceli and Tamás Kovácsházy, "Transients in reconfigurable DSP systems," *IEEE Trans. on Instrumentation and Measurement*, vol. 48, pp. 986–989, October 1999.

[3] Youmin Zhang and Jin Jiang, "Design of integrated fault detection, diagnosis and reconfigurable control system," *in Proceedings of the 38th IEEE Conference on Decision and Control*, Phoenix, Arizona, USA, Dec 1999, vol. 4, pp. 3587–3592.

[4] Tamás Kovácsházy, Gábor Péceli, Gyula Simon, Gabor Karsai, "Realization and Real-time Properties of Reconfiguration and Transient Management Methods," Technical Report, Budapest, Hungary, 2002. http://www.mit.bme.hu/~khazy/publications/tranman_examples_v10.pdf

[5] Linda Walls, Suresh Kannan, Sam Sander, Murat Guler, Bonnie Heck, J.V.R. Prasad, Daniel Schrage, George Vachtsevanos, "An Open Platform for Reconfigurable Control," *IEEE Control Magazine, pp. 49-64, June, 2001*

[6] Hermann Kopetz. Real-Time Systems, Design Principles for Distributed Embedded Applications. Kluwer Academic Publishers, 3300 AH Dordrecht, NL, 1997.

[7] Padmanabhan, M., K. Martin, and G. Péceli, *Feedback-based Orthogonal Filters: Theory, Applications, and Implementation.* Kluwer Academic Publishers, Boston-Dordrecht-London, 1996. 265 p.

[8] V. John Mathews, Giovanni L. Sicuranza, *Polynomial Signal Processing* , John Wiley & Sons, Inc., New York, 2000. 452 p.

[9] Tamás Kovácsházy, Gábor Péceli, Gyula Simon, "Transients in Reconfigurable Signal Processing Channels," *IEEE Trans. on Instrumentation and Measurement*, Vol. 50., pp. 936-940, August, 2001.

[10] Gyula Simon, Tamás Kovácsházy, Gábor Péceli, "Transient Management in Reconfigurable Control Systems," Technical Report, Budapest, Hungary, 2002. http://www.mit.bme.hu/~khazy/publications/recon_techrep.pd

[11] [8] Harel, D., "Statecharts: A Visual Formalism for Complex Systems", *Science of Computer Programming* 8, pp. 231-274, 1987.

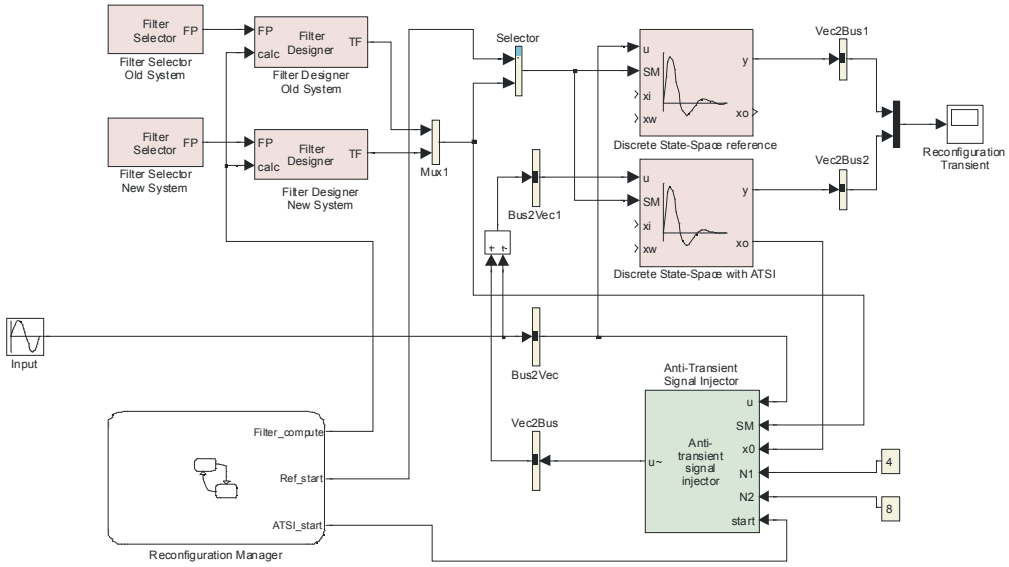

Figure 3 Illustrative example to demonstrate the capabilities of the transient management block library by implementing an anti-transient signal injections scheme

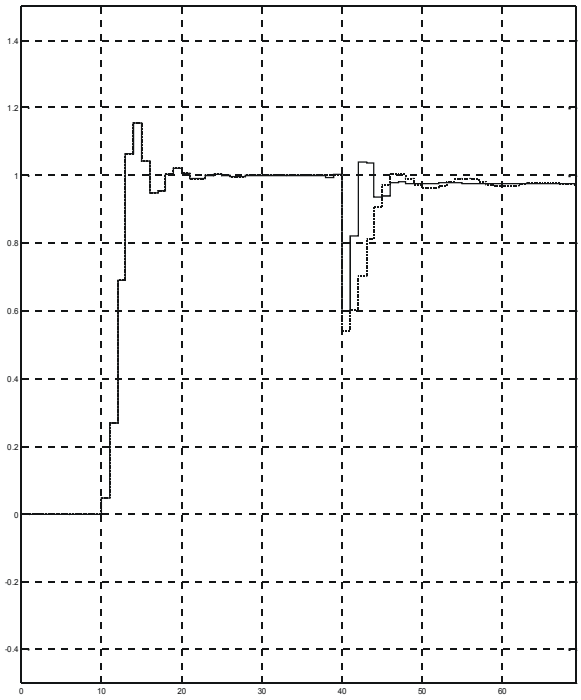

Figure 4 Outputs of the anti-transient signal injection experiment. The reconfiguration occurred at 40s. (solid line: with ATSI, dotted line: without ATSI