

Chapter 2

Modeling, Analysis, and Implementation of Streaming Applications for Hardware Targets

Kaushik Ravindran, Arkadeb Ghosal, Rhishikesh Limaye, Douglas Kim, Hugo Andrade, Jeff Correll, Jacob Kornerup, Ian Wong, Gerald Wang, Guang Yang, Amal Ekbal, Mike Trimborn, Ankita Prasad and Trung N. Tran

Abstract Application advances in the signal processing and communications domains are marked by an increasing demand for better performance and faster time to market. This has motivated model-based approaches to design and deploy

K. Ravindran (✉) · A. Ghosal · R. Limaye · D. Kim · H. Andrade · J. Correll · J. Kornerup · I. Wong · G. Wang · G. Yang · A. Ekbal · M. Trimborn · A. Prasad · T. N. Tran
National Instruments Corporation, Berkeley, CA, USA
e-mail: kaushik.ravindran@ni.com

A. Ghosal
e-mail: arkadeb.ghosal@ni.com

R. Limaye
e-mail: rhishikesh.limaye@ni.com

D. Kim
e-mail: douglas.kim@ni.com

H. Andrade
e-mail: hugo.andrade@ni.com

J. Correll
e-mail: jeff.correll@ni.com

J. Kornerup
e-mail: jacob.kornerup@ni.com

I. Wong
e-mail: ian.wong@ni.com

G. Wang
e-mail: gerald.wang@ni.com

G. Yang
e-mail: guang.yang@ni.com

A. Ekbal
e-mail: amalekbal@ni.com

M. Trimborn
e-mail: mike.trimborn@ni.com

such applications productively across diverse target platforms. Dataflow models are effective in capturing these applications that are real-time, multi-rate, and streaming in nature. These models facilitate static analysis of key execution properties like buffer sizes and throughput. There are established tools to generate implementations of these models in software for processor targets. However, prototyping and deployment on hardware targets, in particular reconfigurable hardware such as FPGAs, are critical to the development of new applications. FPGAs are increasingly used in computing platforms for high performance streaming applications. They also facilitate integration with real physical I/O by providing tight timing control and allow the flexibility to adapt to new interface standards. Existing tools for hardware implementation from dataflow models are limited in their ability to combine efficient synthesis and I/O integration and deliver realistic system deployments. To close this gap, we present the LabVIEW DSP Design Module from National Instruments, a framework to specify, analyze, and implement streaming applications on hardware targets. DSP Design Module encourages a model-based design approach starting from streaming dataflow models. The back-end supports static analysis of execution properties and generates implementations for FPGAs. It also includes an extensive library of hardware actors and eases third-party IP integration. Overall, DSP Design Module is an unified design-to-deployment framework that translates high-level algorithmic specifications to efficient hardware, enables design space exploration, and generates realistic system deployments. In this chapter, we illustrate the modeling, analysis, and implementation capabilities of DSP Design Module. We then present a case study to show its viability as a model-based design framework for next generation signal processing and communications systems.

2.1 Introduction

Dataflow models are widely used to specify, analyze, and implement multi-rate computations that operate on streams of data. Static Dataflow (SDF)¹ is a well known model of computation for describing signal processing applications [1]. An SDF model is a graph of computation actors connected by channels that carry data tokens. The semantics require the number of tokens consumed and produced by an actor per firing be fixed and pre-specified. This guarantees static analyzability of key execution properties, such as deadlock-free operation and bounded memory requirements [2].

A. Prasad
e-mail: ankita.prasad@ni.com

T. N. Tran
e-mail: trung.tran@ni.com

¹ SDF was called *Synchronous* Dataflow in the original works that introduced the model [1, 2]. But the model is fundamentally asynchronous, since actors can fire independently and asynchronously. For this reason, and in order not to confuse SDF with truly synchronous models such as synchronous FSMs, we prefer the term *Static* Dataflow.

The expressiveness of dataflow models in naturally capturing streaming applications, coupled with formal analyzability properties, has made them popular in the domains of multimedia, signal processing, and communications. These high level abstractions are the starting points for model-based design approaches that enable productive design, fast analysis, and efficient correct-by-construction implementations. Ptolemy II [3], Grape-II [4], LabVIEW [5], and Simulink [6] are examples of successful tools built on the principles of model-based design from dataflow models. These tools provide a productive way to translate domain specific applications to efficient implementations.

Most of these tools predominantly deliver software implementations for general purpose and embedded processor targets. However, ever-increasing demands on performance and throughput of new applications and standards have motivated prototyping and deployment on hardware targets, in particular reconfigurable hardware such as Field Programmable Gate Arrays (FPGAs). FPGAs are integral components of modern computing platforms for high performance signal processing. The configurability of FPGAs and constraints of hardware design bring unique implementation challenges and performance-resource trade-offs. FPGAs permit a range of implementation topologies of varying degrees of parallelism and communication schemes. Another requirement for hardware design is the integration of pre-created configurable intellectual property (IP) blocks. Actor models must capture relevant variations in data access patterns and execution characteristics of IP configurations.

Common software synthesis techniques for dataflow models are not immediately applicable to hardware synthesis. Related tools from prior works either make restrictive assumptions about the dataflow models to simplify hardware generation, or do not provide a complete path to implementation on realistic targets supporting established design methodologies and IP libraries. They do not fully address the complexities of connecting hardware actors and IP with differing interfaces. The tools are limited in their ability to deliver realistic system deployments, combining efficient hardware synthesis and physical I/O integration.

In this work, we advance a design framework to enable the design and deployment of streaming applications on FPGA targets. The objective is to empower application and domain experts without specialized knowledge in hardware design to create efficient implementations on FPGA targets. Founded on model-based design principles, the framework emphasizes six key elements to address the hardware design challenge:

- An appropriate model of computation to capture high-level intent.
- An IP library for general computing and domain specific functions.
- A graphical interface that incorporates latest interactive technologies.
- Measurement and control I/O to design and test such systems.
- Static analysis and evaluation of key trade-offs at design time.
- Efficient deployments on high performance hardware platforms.

We combine these elements in the National Instruments (NI) LabVIEW DSP Design Module, a framework for hardware-oriented specification, analysis, and

implementation of streaming dataflow models. The framework enables DSP domain experts to express complex applications and performance requirements in an algorithmic manner and automatically generate efficient hardware implementations. The main components of DSP Design Module are: (a) a graphical specification language to design streaming applications, (b) an analysis engine to validate the model, select buffer sizes and optimize resource utilization to meet throughput constraints, and perform other pertinent optimizations, and (c) implementation support to generate an efficient hardware design and deploy it on Xilinx FPGAs.

The specification is based on the Static Dataflow (SDF) model of computation with extensions to support cyclo-static data rates and parameterization. Prior studies on modeling requirements for wireless communications applications endorse that this model is sufficiently expressive in capturing relevant execution and performance characteristics [7, 8]. DSP Design Module provides an extensive library of math and signal processing functions that harness the resource elements on the FPGA. It also facilitates integration of custom-designed hardware blocks and third-party IP into the design. The back-end eases exploration of design trade-offs and translates a high level algorithmic specification to an efficient hardware implementation.

DSP Design Module enables application domain experts implement complex streaming applications on FPGAs. The hardware implementations are currently targeted to Xilinx Virtex-5 FPGAs on the National Instruments PXIe platform. This is an important distinction from prior works which stop with RTL synthesis from high level models. In contrast, DSP Design Module is integrated with LabVIEW, a commercial framework for deploying systems that combine processors, FPGAs, real-time controllers, and analog and digital I/O. In particular, the tight contract-based integration between DSP Design Module and LabVIEW FPGA enables orthogonalization of design. One designer can focus on the details of integration to I/O via LabVIEW FPGA and control the tight timing requirements of hardware interfaces at the cycle level. Another designer can specify the I/O requirements as equivalent port constraints on a DSP Design Module algorithm description, so that the tool can generate code that satisfies these constraints. DSP Design Module hence delivers a unified design-to-deployment framework for streaming applications.

In this chapter, we highlight salient features of the NI LabVIEW DSP Design Module and illustrate a design flow to implement streaming applications. We then present a case study on the design and implementation of a real-time single antenna Orthogonal Frequency Division Multiplexing (OFDM) wireless communication link transmitter and receiver. The OFDM is part of the Long Term Evolution (LTE) mobile networking standard [9]. The system is implemented on the National Instruments PXIe platform equipped with Xilinx Virtex-5 FPGAs, real-time controllers, and baseband and RF transceivers. We demonstrate the design of key application components in DSP Design Module and show how the analysis capabilities of the tool help the designer make key performance and resource trade-offs. We finally show the deployment of the system on the PXIe platform. The study illustrates how DSP Design Module serves as a model-based design framework that translates high level streaming models to efficient hardware implementations.

2.2 Related Work

Synthesis flows from Register Transfer Level (RTL) logic and behavioral languages (typically C, C++, or SystemC) for hardware targets has been a popular topic of several studies. However, there is limited prior art on hardware generation from non-conventional high level models like dataflow. Ptolemy II is a prominent academic framework for graphical specification and analysis of multiple models of computation [3]. Grape-II facilitates emulation of SDF models on FPGAs and explores memory minimization techniques [4]. While these tools provide some support for RTL generation from restricted models, the focus is more on proof-of-concept and less on optimized hardware implementation.

The work of Edwards and Green also confirms these limitations [10]. They explicitly address hardware generation and evaluate architectures for implementation of SDF models. They also discuss strategies to optimize function parallelism and token buffering on FPGAs. Horstmannshoff et al. [11] and Jung et al. [12] also discuss hardware synthesis techniques from SDF models. They develop analysis methods for resource allocation and schedule computation suitable for hardware. However, these prior works do not consolidate the proposed techniques into a comprehensive design framework that enables implementation on realistic targets. They do not support established synthesis methodologies and IP libraries. Actor definition and IP integration are practical challenges in hardware design, yet these works do not address them adequately.

On the commercial front, LabVIEW FPGA from National Instruments is a popular tool that supports FPGA deployment from dataflow models [13]. However, LabVIEW FPGA only supports the Homogeneous Static Dataflow (HSDF) model of computation, which does not natively capture streaming multi-rate computations. Furthermore, it requires explicit specification of cycle-level behavior for generating efficient hardware. System Generator from Xilinx is another related offering that supports FPGA implementations of synchronous reactive and discrete time models of computation [14]. However, these models are not suitable for data driven streaming specifications that are better expressed in dataflow. SystemVue ESL from Agilent supports more expressive dataflow models and provides libraries and analysis tools for the RF and DSP domains [15]. However, it primarily serves as an exploration and simulation environment, and does not offer a path to implementation in hardware. To our knowledge, there is no integrated tool that addresses the challenges in translating dataflow models to implementations on practical hardware targets.

The closest effort in synthesizing hardware from dataflow programs is the Open Dataflow framework [16]. The CAL actor language in Open Dataflow is an important step in formalizing actor and interface definitions. It has been adopted by the MPEG Video Coding group to develop codecs for future standards [17]. CAL builds on the Dynamic Dataflow model of computation, which allows specifications with variable data rates and non-deterministic behaviors. But this model is undecidable and cannot be subject to static analysis. There have been recent efforts to detect analyzable regions in a CAL model that adhere to restricted models of computation [18].

However, the language does not constrain the designer from introducing structures that might prohibit analysis. In contrast, the DSP Design Module model of computation is intentionally restrictive in order to enable efficient analysis of deadlock-free execution, buffer size, and throughput. While the model may not be as expressive as CAL, it is nevertheless applicable to a range of interesting signal processing applications. Also, CAL is a textual specification language, whereas DSP Design Module is a graphical design environment. We believe a graphical framework built on analyzable models provides an intuitive design approach for domain experts not specialized in hardware design to create efficient system deployments.

In summary, DSP Design Module is an attempt to integrate the respective strengths of the previously discussed tools into a unified design-to-deployment framework for streaming applications on hardware targets. The graphical design environment is intended for algorithm designers who are generally not experts in hardware design. The framework supports analysis capabilities relevant to hardware implementation and includes an extensive library of common math, signal processing, and communications functions. It also enables integration of IPs from native and third-party libraries, like the Xilinx CoreGen library [19], which are essential to practical efficient hardware design. Finally, it delivers implementations on realistic hardware targets that can be combined with other processors, memories, and physical I/O to build high performance embedded systems.

2.3 DSP Design Module: Models and Analysis

The DSP Design Module captures specifications based on the Static Dataflow (SDF) model of computation, with extensions for cyclo-static data rates and parameterization. This section presents the relevant characteristics of these models and illustrates their suitability for specifying signal processing applications.

2.3.1 Static Dataflow

A dataflow model consists of a set of actors (which represent computational units) inter-connected via channels (which communicate data, abstracted as tokens, between actors). In the *Static Dataflow* (SDF) model of computation, at each firing, an actor consumes a fixed number of tokens from each input channel, and produces a fixed number of tokens on each output channel. A channel stores the generated tokens until an actor consumes the tokens. We illustrate the SDF model and relevant properties using the model shown in Fig. 2.1. The model is composed of two actors: *distribute stream*, and *interleave stream*, henceforth referred to as *distribute* and *interleave* actors. The *distribute* actor splits an incoming stream of data into two output streams, while the *interleave* actor merges two incoming streams into one outgoing stream. The *distribute* actor consumes 2 tokens from the incoming channel and produces 1 token on each of the two output channels; the effect of the actor

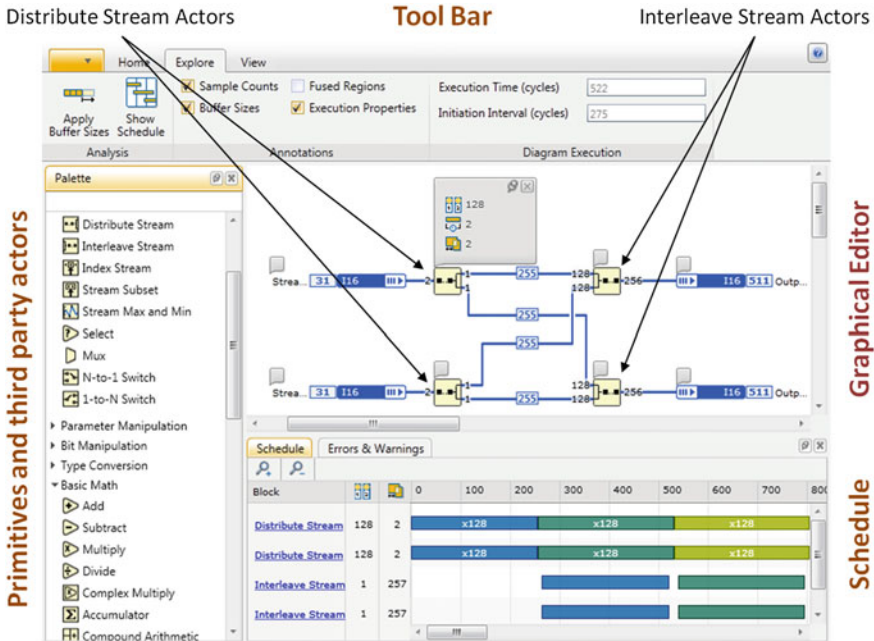


Fig. 2.1 This is the LabVIEW DSP design module graphical user interface in which an SDF model is captured. At the *top* is the tool bar, where the user can set platform characteristics and select different analysis functions. At the *left* is the tool library with different math and signal processing primitives and third party actors. In the *middle* is the graphical canvas where the SDF model is specified. The *lower part* shows the execution schedule along with any errors and warnings

processing is that the first (resp. second) outgoing stream consists of tokens from odd (resp. even) numbered positions of the incoming stream. The *interleave* actor consumes 128 tokens from each of the two incoming channels and produces 256 tokens on the outgoing channel. The actor interleaves the tokens such that the outgoing stream is a concatenation of blocks of 128 tokens taken in alternating fashion from the two input streams. The model has two instances of the *distribute* and *interleave* actors, with each instance of the *distribute* actors connected to both instances of the *interleave* actors. The actors do not change the tokens; they only rearrange the tokens at the outputs so that outgoing streams consist of a different sequence of tokens from the incoming streams.

Each actor is associated with an *execution time* and an *initiation interval*. Execution time is the time (in clock cycles) that the actor needs to process inputs, perform computation, and generate outputs. Initiation interval is the minimum time (in clock cycles) between consecutive firings of an actor. If initiation interval is less than execution time for an actor, the actor may fire in an overlapping (pipelined) fashion. In the above example, the execution time and the initiation interval of the *distribute* actor are 2 cycles each; the execution time and the initiation interval of the *interleave* actor are 257 cycles each.

2.3.2 SDF: Properties and Analysis

The SDF model of computation permits decidability of key properties of streaming applications, such as bounded memory and deadlock free execution [1]. A model is bounded if it can be executed without termination using buffers with finite capacity. An execution of an SDF model can achieve periodic behavior because the token consumption and production rates are constant. In SDF models, bounded execution is verified by proving that the model is *sample rate consistent* [2]. An SDF model is sample rate consistent if there exists a fixed non-zero number of firings for each actor such that executing these firings reverts the model to its original state. A vector denoting this number for each actor is called the *repetitions vector*. Consider a channel between any *distribute* and *interleave* actors in Fig. 2.1. Given that the *distribute* actor produces 1 token and the *interleave* actor consumes 128 tokens, the *distribute* actor needs to fire 128 times more often than the *interleave* actor in each round of firing. Hence, the repetitions count for the *distribute* actors is 128, while that for the *interleave* actors is 1. These repetition counts ensure that the number of tokens produced over a channel is equal to the number of tokens consumed. This in turn guarantees that any non-terminating periodic execution of the SDF model can be implemented with buffers that have bounded capacity.

In addition to boundedness, deadlock-free operation is essential for most streaming applications. Dataflow models that are bounded and deadlock-free are viable for implementation in hardware and software. An SDF model is *deadlock-free* if at any point in its execution, there is at least one actor that can fire. For a sample rate consistent SDF model, it suffices to verify that a single *iteration* (in which each actor fires as many times as specified in the repetitions vector) of the model can be completely executed. The solution is to compute a *self timed schedule* (in which an actor fires as soon as all input tokens are available) for one iteration [2]. As the model is consistent, this schedule can be executed infinitely often. This computation assumes that channels in the SDF model are implemented as FIFO buffers with infinite capacity. Nevertheless, a bound on the buffer size can be encoded as a back edge in the SDF model with initial tokens corresponding to the size of the buffer [20]. The model in Fig. 2.1 is bounded and deadlock free.

Important performance metrics for real-time streaming applications are throughput (i.e., rate of tokens produced by an actor) and memory (i.e., buffer sizes to implement channels). Each actor needs to wait until the respective producing actor has generated the required number of tokens, which imposes restrictions on the minimum buffer sizes required to implement the channels. For the example model in Fig. 2.1, the *interleave* actor needs 128 tokens per firing, whereas the *distribute* actor produces 1 token per firing, i.e., one firing of the *interleave* actor depends on 128 firings of the *distribute* actor. Hence any channel between the two actors must accommodate at least 128 tokens. Static analysis can also determine the buffer sizes of the channels needed to meet the throughput requirements of the model. Buffer sizes are dictated by the data rates on the channels, the throughput requirements, and the clock rate at which the model is executed. Throughput is usually specified in

engineering units such as Mega-Samples per second (MSps). Setting the throughput at the outputs as 10 MSps for a clock rate of 40 MHz in the example model, the buffer size required for each channel is 255; the buffer size can achieve a maximum throughput of 37.23 MSps for the clock rate of 40 MHz. The *schedule* (i.e., order of firings) of the actors captures how the actors are executed in time. The schedule for the example (shown at the bottom of Fig. 2.1) shows that the *distribute* and *interleave* actors fire in parallel, with the *distribute* actors firing 128 times more often than the *interleave* actors.

2.3.3 Extensions for Cyclo-Static Data Rates and Parameterization

The SDF model of computation accurately captures fixed-length computations on streams. But there are some actors that follow a pre-specified cyclic pattern in the number of tokens processed. For such computations, the *Cyclo-Static Dataflow* (CSDF) [21] model of computation generalizes SDF by allowing the number of tokens consumed or produced by an actor to vary according to a fixed cyclic pattern. Each firing of a CSDF actor corresponds to a phase of the cyclic pattern. In Fig. 2.1, if the input token count of the *interleave* actors is replaced by a cyclic pattern (64, 128, 256), then the result is a CSDF model that interleaves incoming streams following a deterministic cyclic pattern of input token counts. As the cyclic pattern is fixed and known at design time, all static analysis properties of SDF are also applicable [22]. We refer to [20, 23] for details on analysis algorithms to compute buffer sizes and throughput.

SDF and CSDF are static in nature. However, some applications may require that the number of tokens processed by an actor vary at run-time. For example, consider a variant of the *interleave* actor that can be configured to consume 64, 128, or 256 tokens on its input channels per firing. The token consumption count on its input channels is no longer a fixed value; it is denoted by a parameter N , where N is from the set {64, 128, 256}. Note that this behavior is different from the CSDF actor discussed earlier. In successive firings, the CSDF actor consumes {64, 128, 256} in a cyclical pattern. In contrast, the parameterized actor consumes N tokens per firing, but the value the parameter is determined by external input and need not follow a pre-specified pattern. The *Parameterized Static Dataflow* (PSDF) model of computation is an extension to SDF that allows such parameterized actors [24].

The PSDF model can be viewed as a composition of several SDF models. At any point in execution, the behavior of a PSDF model corresponds to the underlying SDF model with all parameters fixed to a specific value. However, if the parameter values are allowed to change in any arbitrary manner, the problem of verifying whether the PSDF model is bounded and deadlock-free becomes undecidable. One useful restriction to facilitate static analysis is to allow changes in parameter values to take effect only at iteration boundaries. This restriction ensures that a PSDF model can be statically analyzed to verify bounded deadlock-free execution by verifying that

the individual SDF models corresponding to different combinations of parameter values are bounded and deadlock-free. Subsequent analysis of throughput and buffer sizes can be computed as the worst case from among all combinations of parameter values. The CSDF model can similarly be parameterized to form the *Parameterized Cyclo-Static Dataflow* (PCSDF) model.

2.4 DSP Design Module: Implementation Flow

DSP Design Module is a graphical environment tool which allows user to specify streaming applications, explore optimizations, and generate synthesizable FPGA designs. Fig. 2.2 summarizes the design and implementation flow in DSP Design Module. In this section we discuss this flow using an OFDM transmitter application as a driving example.

2.4.1 Design Environment

The user works in a graphical environment as shown in Fig. 2.1. The starting point is the *Application*, e.g. a DSP algorithm, which the user starts drawing by selecting actors from the *Actor Library* and placing them on the editor canvas. This begins the

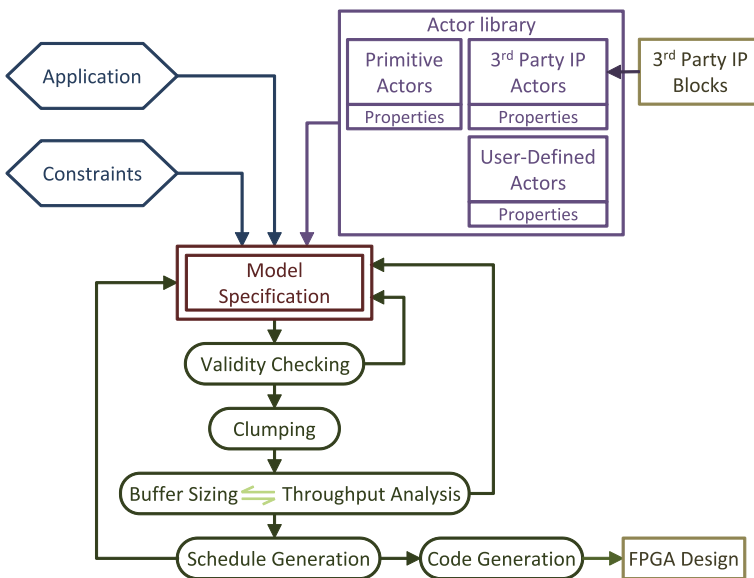


Fig. 2.2 Design and implementation flow in DSP design module

Model Specification step. The actor library consists of a rich set of primitive actors, including computational actors (add, multiply, square root, sine, log etc.), stream manipulation actors (upsample, downsample, distribute stream, interleave stream, etc.), third-party actors (e.g. FFT and FIR blocks from Xilinx Coregen [19]), and user-defined actors that are either specified in the LabVIEW programming language or previously defined models constructed using DSP Design Module. This reuse of actors allows for hierarchical composition of designs within the tool.

Figure 2.3 shows the SDF model for the OFDM Transmitter. The basic building blocks of the transmitter consist of data and reference symbol interleaving and zero padding the signal, followed by FFT processing and sample rate conversion. The input signals are first interleaved and distributed followed by zero padding. The streams are then processed by an FFT actor, and subsequently down sampled using two FIR filters. The FFT (Xilinx 7.0) and FIR (Xilinx 5.0) filters are third party actors. The other actors are primitives of the DSP Design Module. The user can also define actors using DSP Design Module. Figure 2.4 shows the same model where the computations for zero padding (performed by the distribute stream, the interleave stream, and the split number actors in Fig. 2.3) are grouped under an actor named ZeroPad (the first actor from left following the input ports).

The user describes the models by connecting the actors, and optionally configures their properties; e.g., FIR filter actors can be configured for different down-sampling/up-sampling scenarios. Configurable properties of an actor include the data types and the number of tokens for the input and output channels. For some actors, throughput, pipeline depth, resource usage, or other implementation-specific options can also be configured by the user. The actor library includes cycle-accurate characteristics for each actor configuration, including initiation interval and execution time.

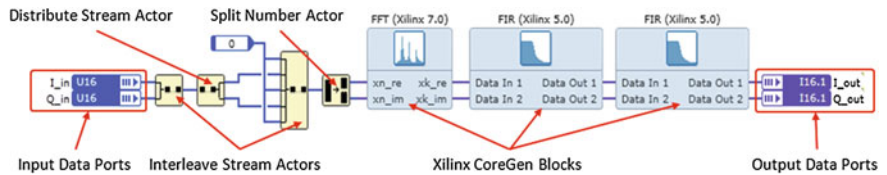


Fig. 2.3 SDF model of the OFDM transmitter

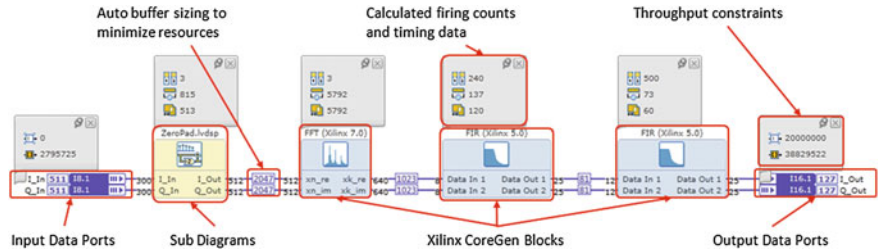


Fig. 2.4 Analysis results for the SDF model of the OFDM transmitter

In the OFDM example, the first FIR filter is configured with a user-defined coefficient file, and the interpolation and decimation factors are set to 25 and 8, respectively. The second FIR filter is configured with another user defined coefficient file, and the interpolation and decimation factors are set to 25 and 12, respectively.

The second input from the user is the *Constraints* for hardware generation. This includes minimum throughput requirements and target clock frequency. Throughputs can be associated with input/output ports or internal channels of the design, and are specified in engineering units, such as Mega-Samples per second (MSps). The model in Fig. 2.4 sets the minimum required throughput at the output ports to be 20 MSps and the target frequency to be 40 MHz.

The tool performs several types of analysis on the design in the background while the user is constructing it, with immediate feedback on the current state of the design. *Validity Checking* verifies bounded and deadlock-free execution of the model. It also performs automatic type checking and type propagation across the design. Errors or warnings are immediately annotated on the offending nodes on the canvas and reported under the Errors and Warning tab in the tool. On a valid design, the tool performs *Clumping* to identify homogeneous and simple regions that fit specialized implementation schemes (see Sect. 2.4.3). *Buffer Sizing* and *Throughput Analysis* are then performed on the design. This determines the buffer sizes required on the channels to satisfy user constraints such as minimum throughput. If the constraints cannot be met, the tool reports errors. *Schedule Generation* establishes a valid, cycle-accurate schedule (Fig. 2.5) for the design, given the determined buffer sizes and clumped regions. Thus the tool provides instant feedback on the run-time behavior of the design, including the achievable throughput.

The user can simulate the functional behavior on the development platform before invoking the hardware implementation stage. As a part of the simulation, the user can specify stimulus data and add graphical displays to the design to visualize the response on output ports or on any wire in the design.

The final step is *Code Generation* that uses the results of analysis to emit an FPGA design in the form of synthesizable LabVIEW files. The tool can also generate a synthesizable testbench that allows the user to stimulate the design from the development computer and compare the response to validated signals. The testbench includes the necessary code for DMA communication between the FPGA device and the development computer. The LabVIEW files can be used to generate a bitfile used to implement the design on Xilinx FPGA devices or for timing-accurate hardware simulation. Currently the tool supports targeting Virtex 5 devices from Xilinx.

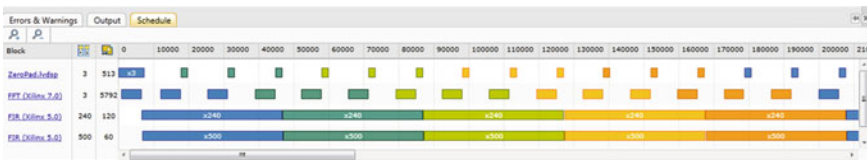


Fig. 2.5 Schedule for the SDF model of the OFDM transmitter

2.4.2 Implementation Strategy

DSP Design Module uses a FIFO-based, *self-timed* implementation strategy to realize the designs on FPGA fabrics [23]. In the FIFO-based strategy every channel in a model is conceptually mapped to a hardware FIFO of appropriate size and every actor is mapped to a dedicated hardware block that implements its functionality. There is no resource sharing among two different channels or two different actors in the current state of the tool, but the model does not preclude this. In the self-timed implementation strategy every actor is fired whenever it has a sufficient number of tokens on each of its input channels, sufficient number of vacancies on each of its output channels, and the initiation interval of the previous firing has expired. This evaluation is done for an actor on every clock cycle, and is independent of the state of the other actors in the model. As a consequence, there is no global scheduling logic in this implementation strategy, reducing the complexity of the controller for each actor in the design.

2.4.3 Glue Design and IP Integration

The FIFO-based, self-timed implementation strategy surrounds each actor with a harness that provides a FIFO interface to realize the SDF model and its extensions discussed in Sect. 2.3. The generated code for an actor presents a standardized interface to the harness, with designated lines for data and handshaking signals. A standardized harness simplifies *glue design*, i.e., the design of communication and control logic to stitch hardware blocks together. Further, a standardized harness provides a mechanism to integrate native and third-party IP. The IP interfaces can be adapted to the harness template, which facilitates the generation of the *glue* to connect them to actors in the model.

A faithful realization of the SDF model of computation requires extra resources for the harness logic and the FIFOs on each channel. However, in the synthesized design this overhead can be significant compared to the resource usage of the actors themselves. To reduce this overhead the tool applies a series of *clumping* transformations to reduce the number of harnesses and FIFOs in the design. These transformations preserve the observable flow of tokens on the input and output ports, while preserving or increasing throughput. A clump is either a single actor or a collection of neighboring actors connected using direct locally synchronized connections without a FIFO-based harness interface. A clump exports a standard actor interface at its boundary. Hence it is a candidate for further clumping with other actors or clumped regions.

The supported clumping transformations are *Simple*, *Combinational*, and *Homogeneous* clumps. In a simple clump, design constants and downstream branches are merged with connected actors, resulting in a clump where the constant inputs are replaced with directly wired constants and the branches translate directly to branched

output wires. Combinational clumps are constructed explicitly in the tool when the user selects connected actors in the design that can complete their entire execution in a single cycle, and designates them as executing jointly in a single cycle. The result is a clump with direct wire connections between the synthesized actors. Homogeneous clumps are regions where all actors are homogeneous, i.e., all the input and output counts are one. These actors are connected in a synchronous fashion, using registers to equalize delays along each path and by using a simple state machine to throttle the inputs to the clump, based on the actor with the greatest initiation interval. The clumping transformation is akin to the process of converting an asynchronous design, where all actors are connected by FIFOs, into a Globally Asynchronous Locally Synchronous architecture (GALS) [25], where FIFOs connect regions of synchronously connected actors called *clumps*.

2.4.4 I/O Integration

As discussed in the introduction, there is a tight contract-based integration between DSP Design Module and LabVIEW FPGA, where the actual I/O takes place. Logically, however, we can view I/O operations as being proxied to DSP Design Module via constraints on ports of the actor diagram, that describe the maximum rate at which data is going to be fed from I/O or other FPGA IP modules to the input ports of a DSP diagram, and the minimum rate required at the output ports of the DSP diagram, to feed the I/O or other FPGA IP modules. This separation of concerns allows the LabVIEW FPGA designer to focus on timed behavior, i.e., details of interacting with the I/O, where the effects of time are clearly observable, while the DSP designer can focus on the (untimed) algorithm and let the tools help with the interaction with the timed boundary to LabVIEW FPGA. In the future, we intend to bring the I/O actors directly onto the DSP diagram as special actors that the user can configure, and automatically modify the properties to propagate the constraints onto the surrounding terminals.

2.5 OFDM Transmitter and Receiver Case Study

In this section, we present a case study on the design and implementation of a real-time single antenna OFDM transmitter and receiver using LabVIEW DSP Design Module. The single antenna OFDM link design is based upon the LTE standard [9] with system specifications that include a transmission bandwidth of 5 MHz, 7.68 MSps sampling rate, 512 FFT length, 128 cyclic prefix (CP) length (extended mode), 250 data subcarriers, 50 reference subcarriers, and variable 4/16/64 Quadrature Amplitude Modulation (QAM). The proposed communication system is implemented on an NI PXI Express platform shown in Fig. 2.6, where the transmitter and receiver consist of the following four main components: (a) *PXIe-8133* Real-time controller

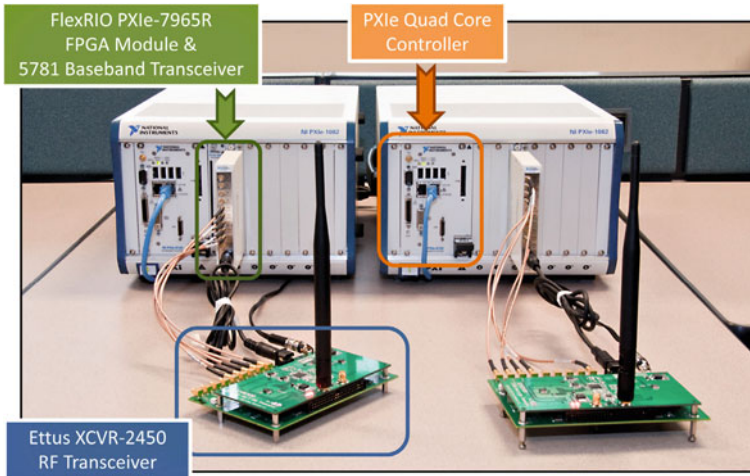


Fig. 2.6 NI PXI Express Real-Time Signal Processing Platform with Ettus Research RF front-end

equipped with a 1.73 GHz quad-core Intel Core i7-820 processor; (b) *PXIe-7965R* FPGA module with a Virtex-5 SX95T FPGA; (c) *NI-5781* 40 MHz baseband transceiver module; and (d) *Ettus Research XCVR-2450* 802.11a/b/g compliant 40 MHz RF transceiver.

Figure 2.7 shows the transmitter and receiver block diagram of the various signal processing tasks. The figure also shows a mapping of the various blocks to the underlying hardware targets and the respective design tools used in their implementation; e.g., the transmitter *Data Bit Generation* block (programmed using LabVIEW Real-time) executes on the PXIe-8133 real-time controller, while the higher rate 512 *IFFT with 128 CP Insertion* block (programmed using DSP Design Module) executes on the PXIe-7965R FPGA module. The various data rates associated with the inputs and

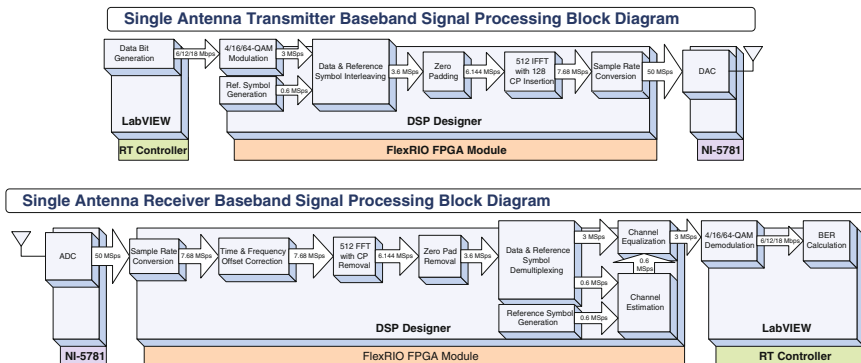


Fig. 2.7 Hardware and software mapping of Transmitter and Receiver block diagrams

outputs of each block are also shown; e.g., the transmitter *Sample Rate Conversion* block up-samples input data streaming at 7.68–50 MSps in order to meet the sample rate constraints of the NI-5781 DAC.

2.5.1 Transmitter and Receiver Overview

We provide an overview of how DSP Design Module implements the transmitter and receiver; refer to [26] for a detailed discussion. In the transmitter, random bytes of data generated by a real-time controller are sent to the FPGA module for Multilevel QAM (M-QAM) [27]. Depending upon the modulation order value, the bytes of data are unpacked into groups of 2, 4, or 6 bits corresponding to 4/16/64-QAM, respectively which are then mapped to their respective complex symbols. After QAM modulation, 250 data symbols are interleaved with 50 reference symbols stored in a look-up table forming an array of 300 interleaved symbols which is then split into two equal groups and padded with zeros forming an array of 512 samples. The 512 samples are passed through a 512 point IFFT block translating the frequency domain samples into the time domain. A 128 point CP is also inserted such that the output of the block consists of 640 samples streaming at 7.68 MSps which is then converted to 50 MSps using two sets of FIR filters. The samples are forwarded to the NI-5781 for digital-to-analog conversion followed by RF up-conversion.

The receiver begins with sample rate down-conversion of the incoming 50 MSps signal (from the ADC) to 7.68 MSps. This is followed by time and carrier frequency offset (CFO) estimation using the blind estimation technique proposed in [28]. After CFO correction, the received OFDM symbol is passed on for CP removal and FFT transformation returning the signal to the frequency domain. Zero pads are then removed, and the reference and data symbols are separated in a de-interleave operation; the data is subsequently processed for channel estimation and channel equalization. Once channel equalization is complete, the data symbol estimates are transferred to the real-time controller at 3 MSps for QAM demodulation and bit error rate calculation.

2.5.2 Hardware Implementation

In addition to the portions of the design implemented in the DSP Design Module, the compilation results include nominal logic implemented in LabVIEW FPGA that manages data transfer across the NI-5781 baseband transceiver and PXIe-7965R FPGA module, and the PXIe-7965R FPGA module and PXIe-8133 controller which is running LabVIEW Real-time. The results also include additional logic to control the NI-5781 module including ADC/DAC read/write operations, sampling frequency configurations, and clock selections.

Table 2.1 shows the summary of the compiled FPGA resource utilization. The first two columns show the various resources available on the PXIe-7965R’s Virtex-5 SX95T FPGA and the total number of elements associated with each resource; the percentage utilization by the transmitter and receiver modules are listed in the last two columns. Note that the receiver uses more than twice the registers and LUTs than the transmitter due to significant difference in computational complexity. The DSP diagrams for the transmitter and receiver are configured for 125 MHz clocks, and both successfully met timing during compilation. Figure 2.8 is a screen shot of the OFDM receiver front panel taken during an over-the-air test of the communications link. In addition to carrier frequency, modulation order, and LNA gain controls, a sample 16-QAM signal constellation plot is shown along with two average bit error rate (BER) curves, one taken on a single subframe basis (lower right hand plot), and the other taken over all received subframes (upper right hand plot). The average BER over all subframes converges to an approximate value of 8×10^{-4} , which is viable for a practical implementation.

2.5.3 Design Exploration

Furthermore, the designer can use the analysis framework to explore trade-offs between the throughput and buffer resources used by the implementation. Table 2.2

Table 2.1 Resource utilization of the OFDM transmitter and receiver on a virtex-5 SX95T FPGA

| Resource name | Available elements | Transmitter utilization (%) | Receiver utilization (%) |
|-----------------|--------------------|-----------------------------|--------------------------|
| Slices | 14720 | 43.1 | 79.2 |
| Slice registers | 58880 | 21.6 | 54.6 |
| Slice LUTs | 58880 | 24.7 | 57.3 |
| DSP48s | 640 | 2.7 | 8.3 |
| Block RAM | 244 | 8.2 | 19.7 |

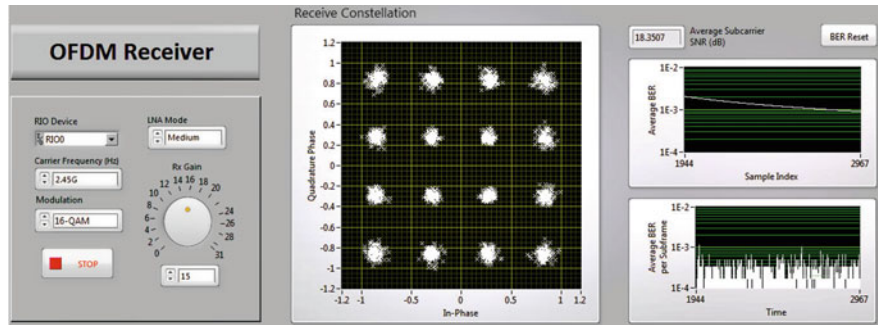


Fig. 2.8 OFDM receiver front panel

Table 2.2 Throughput and buffer size trade-offs for the OFDM transmitter and receiver models

| Name | #Actors, #channels | Firings/ iteration | Throughput (MSps) | BlockRAM utilization (%) | Run-time (seconds) |
|---------------------|-----------------------|-----------------------|----------------------|-----------------------------|-----------------------|
| OFDM Transmitter | 1114 | 6250 | 2 | 3.6 | 0.7 |
| | | | 5 | 4.1 | 1 |
| | | | 25 | 6.4 | 1 |
| | | | 50 | 8.2 | 2 |
| OFDM Receiver | 4666 | 25000 | 15 | 3.1 | 40 |
| | | | 50 | 19.7 | 39 |

summarizes these trade-offs for the OFDM transmitter and receiver models. The table reports the number of actors, channels, and firings per iteration for each model. The target throughputs are specified in Mega-Samples-per-second (MSps). The buffer resources required by the design to meet the target throughput are specified in terms of the percentage utilization of the BlockRAM on the PXIe-7965R's Virtex-5 SX95T FPGA target. The CPU run time for the analysis on an Intel Core i7 2.8 GHz processor is also reported.

As expected, lower target throughputs require fewer buffer resources. The graphical environment allows the designer to quickly explore different design configurations and inspect the resource utilization and schedule without generating full hardware implementations. The run times for the more complex receiver models is still less than a minute, which is reasonable for an interactive exploration framework.

2.5.4 Extensions

Following on from these results, we have been able to demonstrate enhancements in resource utilization and performance by using more detailed actor information, in particular data access patterns for SDF actors. The SDF model is limited in its ability to specify how data is accessed in time which often leads to sub-optimal implementations using more resources than necessary. An extension to the SDF model, called Static Dataflow with Access Patterns (SDF-AP), overcomes this limitation by including access patterns which captures the precise time when tokens are read and written at ports. The SDF-AP model retains the analyzability of SDF-like models while accurately capturing the interface timing behavior. References [29, 30] introduce the theory behind the SDF-AP model of computation and demonstrate throughput and resource usage improvements in resulting hardware implementations. [31] presents analysis methods for key execution properties (boundedness, deadlock, throughput, and buffer size) of the SDF-AP model. Experimental results indicate that the SDF-AP model can reduce the buffer size requirements of the OFDM application discussed earlier by about 63 %. We have developed a research prototype that extends DSP Design Module to generate hardware implementations from SDF-AP

models. Even though the DSP Design Module application language is expressive enough to capture a variety of communication applications, there are some applications that could benefit from additional modal behavior. To that effect, we are also studying extensions to the application language based on Heterochronous Dataflow (HDF) [32] and Scenario-Aware Data Flow (SADF) [33].

2.6 Summary

In this article, we presented National Instruments LabVIEW DSP Design Module, a framework to specify dataflow models of streaming applications, analyze them, and generate efficient implementations for hardware targets. The Static Dataflow model with extensions for cyclo-static data rates and parameterization is sufficiently expressive in specifying high level streaming applications that underlie today's complex communications systems. The back-end performs key optimizations related to buffer sizing and scheduling. The framework has a rich actor library and facilitates integration of custom-designed IPs from native and third-party sources. Thus, LabVIEW DSP Design Module serves as a system design and exploration framework that enables domain experts, who are not specialized in hardware design, to productively specify applications using high level models of computation and still create realistic deployments, combining efficient hardware synthesis and physical I/O integration. The case study of the OFDM transmitter and receiver shows its viability as a model-based design framework for next generation signal processing and communications systems. Recent advances related to the specification of access patterns and modal behavior in dataflow models are promising extensions to this work.

References

1. Lee, E.A., Messerschmitt, D.G.: Synchronous data flow. *Proc. of the IEEE* **75**(9), 1235–1245 (1987)
2. Bhattacharyya, S.S., Murthy, P.K., Lee, E.A.: *Software Synthesis from Dataflow Graphs*. Kluwer Academic Press, Norwell (1996)
3. Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity: the Ptolemy approach. *Proc. of the IEEE* **91**(1), 127–144 (2003)
4. Lauwereins, R., Engels, M., Adé, M., Peperstraete, J.A.: Grape-II: a system-level prototyping environment for DSP applications. *Computer* **28**(2), 35–43 (1995)
5. Andrade, H.A., Kovner, S.: Software synthesis from dataflow models for G and LabVIEW. In: *Proceedings of the IEEE Asilomar conference on signals, systems, and computers*, pp. 1705–1709 (1998)
6. The MathWorks Inc.: *Simulink user's guide* (2005). <http://www.mathworks.com>
7. Kee, H., Shen, C.C., Bhattacharyya, S.S., Wong, I., Rao, Y., Kornerup, J.: Mapping parameterized cyclo-static dataflow graphs onto configurable hardware. *J. Signal Process. Syst.* vol. **66**, pp. 285–301 (2012). doi:[10.1007/s11265-011-0599-5](https://doi.org/10.1007/s11265-011-0599-5)

8. Berg, H., Brunelli, C., Lücking, U.: Analyzing models of computation for software defined radio applications. In: International symposium on system-on-chip (SOC), pp. 1–4. Tampere, Finland (2008)
9. 3GPP LTE: The mobile broadband standard (2008) <http://www.3gpp.org/>
10. Edwards, M., Green, P.: The implementation of synchronous dataflow graphs using reconfigurable hardware. In: Proceedings of FPL '00, pp. 739–748 (2000)
11. Horstmannshoff, J., Meyr, H.: Optimized system synthesis of complex RT level building blocks from multirate dataflow graphs. In: Proceedings of ISSS '99, pp. 38–43 (1999)
12. Jung, H., Lee, K., Ha, S.: Optimized RTL code generation from coarse-grain dataflow specification for fast HW/SW cosynthesis. *J. Signal Process. Syst.* **52**(1), 13–34 (2008)
13. National Instruments Corp.: LabVIEW FPGA. <http://www.ni.com/fpga>
14. Xilinx Inc.: System generator for DSP: getting started guide. <http://www.xilinx.com>
15. Hsu, C.J., Pino, J.L., Hu, F.J.: A mixed-mode vector-based dataflow approach for modeling and simulating the physical layer. In: Proceedings of the 47th design automation conference, DAC '10, pp. 18–23. ACM, New York, USA (2010)
16. Janneck, J.W.: Open dataflow (OpenDF). <http://www.opendf.org/>
17. Janneck, J., Miller, I., Parlour, D., Roquier, G., Wipliez, M., Raulet, M.: Synthesizing hardware from dataflow programs: an MPEG-4 simple profile decoder case study. In: IEEE workshop on signal processing systems, pp. 287–292 (2008)
18. Gu, R., Janneck, J.W., Raulet, M., Bhattacharyya, S.S.: Exploiting statically schedulable regions in dataflow programs. In: Proceedings of the 2009 IEEE international conference on acoustics, speech and signal processing, ICASSP '09, pp. 565–568. IEEE Computer Society, Washington, USA (2009)
19. Xilinx Inc.: Xilinx core generator, ISE design suite 12.1. Xilinx Inc. (2010)
20. Stuijk, S., Geilen, M., Basten, T.: Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In: Proceedings of DAC '06, pp. 899–904 (2006)
21. Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J.A.: Cyclo-static dataflow. *IEEE Trans. Signal Process.* **44**(2), 397–408 (1996)
22. Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J.: Cyclo-static data flow. In: IEEE international conference acoustics, speech, and signal processing. vol. 5, pp. 3255–3258 (1995)
23. Moreira, O.M., Bekooij, M.J.G.: Self-timed scheduling analysis for real-time applications. *EURASIP J. Adv. Signal Process.* **2007**(83710), 1–15 (2007)
24. Bhattacharya, B., Bhattacharyya, S.: Parameterized dataflow modeling for DSP systems. *IEEE Trans. Signal Process.* **49**(10), 2408–2421 (2001)
25. Chapiro, D.M.: Globally-asynchronous locally-synchronous systems. Ph.D. thesis, Stanford University, CA (1984)
26. Andrade, H., Correll, J., Ekbal, A., Ghosal, A., Kim, D., Kornerup, J., Limaye, R., Prasad, A., Ravindran, K., Tran, T.N., Trimborn, M., Wang, G., Wong, I., Yang, G.: From streaming models to FPGA implementations. In: Proceedings of the conference for engineering of reconfigurable systems and algorithms (ERSA-IS). Las Vegas, USA (2012)
27. Proakis, J.: Digital communications, 4th edn. McGraw-Hill Science/Engineering/Math (2000)
28. Sandell, M., van de Beek, J.J., Brjesson, P.O.: Timing and frequency synchronization in OFDM systems using the cyclic prefix. In: Proceedings of international symposium synchronization, pp. 16–19 (1995)
29. Ghosal, A., Limaye, R., Ravindran, K., Tripakis, S., Prasad, A., Wang, G., Tran, T.N., Andrade, H.: Static dataflow with access patterns: semantics and analysis. In: Proceedings of the 49th annual design automation conference, DAC '12, pp. 656–663. ACM, New York, USA (2012)
30. Tripakis, S., Andrade, H., Ghosal, A., Limaye, R., Ravindran, K., Wang, G., Yang, G., Kornerup, J., Wong, I.: Correct and non-defensive glue design using abstract models. In: Proceedings of the seventh IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis, CODES+ISSS '11, pp. 59–68. ACM, New York, USA (2011)
31. Ravindran, K., Ghosal, A., Limaye, R., Wang, G., Yang, G., Andrade, H.: Analysis techniques for static dataflow models with access patterns. In: Proceedings of the 2012 conference on design and architectures for signal and image processing, DASIP '12 (2012)

32. Girault, A., Lee, B., Lee, E.A.: Hierarchical finite state machines with multiple concurrency models. *IEEE Trans. Comput. Aided Des.* **18**(6), 742–760 (1999)
33. Theelen, B.D., Geilen, M.C.W., Basten, T., Voeten, J.P.M., Gheorghita, S.V., Stuijk, S.: A scenario-aware data flow model for combined long-run average and worst-case performance analysis. In: *Proceedings of MEMOCODE'06*, pp. 185–194 (2006)

Embedded Systems Development

From Functional Models to Implementations

Sangiovanni-Vincentelli, A.; Zeng, H.; Di Natale, M.;

Marwedel, P. (Eds.)

2014, VIII, 223 p., Hardcover

ISBN: 978-1-4614-3878-6