From Streaming Models to FPGA Implementations

ERSA'12 Industrial Regular Paper

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

***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 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, such as FPGAs, are critical to the development of new applications. FPGAs are increasingly used in computing platforms for high performance streaming applications. Existing tools for hardware implementation from dataflow models are limited in their capabilities. To close this gap, we present DSP Designer, a framework to specify, analyze, and implement streaming applications on hardware targets. DSP Designer encourages a model-based design approach starting from a Parameterized Cyclo-Static Dataflow model. 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 Designer is an exploration framework that translates high-level algorithmic specifications to efficient hardware. In this paper, we illustrate the modeling, analysis, and implementation capabilities of DSP Designer. Through a detailed case study, we show that DSP Designer is viable for the design of next generation signal processing and communications systems.**

I. Introduction

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

Over the years, several extensions of SDF have been developed that improve the expressiveness of the model while preserving decidability, such as Cyclo-Static Dataflow (CSDF) [3], Parameterized Static Dataflow (PSDF) [4], Heterochronous Dataflow (HDF) [5], Scenario-Aware Dataflow (SADF) [6], and Static Dataflow with Access Patterns (SDF-AP) [7]. Complementing these modeling advances, algorithmic solutions for static analysis have been studied in depth. Viable techniques have been developed for computation of throughput, buffer sizes, and schedules [2] [8] [9].

The expressiveness of dataflow models in naturally capturing streaming applications, coupled with formal compile time 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 [10], LabVIEW [11], and Simulink [12] are examples of successful tools built on the principles of model-based design from dataflow models.

These tools predominantly deliver software implementations for general purpose and embedded processor targets. However, ever-increasing demands on performance of new applications and standards have motivated prototyping and deployment on hardware targets, such as Field Programmable Gate Arrays (FPGAs). FPGAs are integral components of modern computing platforms for high performance signal processing. Surprisingly, few studies have been directed to the synthesis of efficient hardware from dataflow models.

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. Fine-grained specification of actor execution at the cycle level enables execution choices between fully specified static schedules and more flexible self-timed schedules. Communication between actors could be through direct wires, handshake protocols, shift registers, shared registers with scheduled access, or dedicated FIFO buffers. Each mechanism poses different requirements on the interface and glue logic to stitch actors. Finally, a key requirement for hardware design is the integration of pre-created configurable intellectual property (IP) blocks. Hardware actor models must capture relevant variations in data access patterns and execution characteristics of different configurations.

We address these challenges with DSP Designer, a framework for hardware-oriented specification, analysis, and implementation of streaming dataflow models. The intent is to enable DSP domain experts to express complex applications and performance requirements in algorithmic manner and to auto-generate efficient hardware implementations. The main components of DSP Designer 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 Parameterized Cyclo-Static Dataflow (PCSDF) model of computation, which

is a sufficiently expressive model for wireless communications applications [13] [14]. DSP Designer 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. Thus, DSP Designer simplifies the creation of complex streaming applications targeted for FPGA deployment.

In this paper, we highlight salient features of DSP Designer and illustrate a design flow to implement streaming applications. We then present a case study on the deployment of an Orthogonal Frequency Division Multiplexing (OFDM) wireless communication link from the Long Term Evolution (LTE) [15] mobile networking standard on a Xilinx FPGA.

II. Related Work

Synthesis flow 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 dataflow models [10]. 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.

On the commercial front, LabVIEW FPGA from National Instruments is a popular tool that supports FPGA deployment from dataflow models [16]. However, LabVIEW FPGA only supports the Homogeneous Static Dataflow (HSDF) model of computation, which does not allow native specification of streaming multi-rate computations. System Generator from Xilinx is another related offering that supports FPGA implementations of synchronous reactive and discrete time models of computation [17]. However, these models are not suitable for data driven streaming specifications. SystemVue ESL from Agilent supports more expressive dataflow models and provides libraries and analysis tools for the RF and DSP domains [18]. However, it primarily serves as an exploration and simulation environment, and does not offer a path to implementation in hardware.

The closest effort in synthesizing hardware from dataflow programs is the Open Dataflow framework [19]. The CAL actor language supported 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 [20]. CAL builds on the Dynamic Dataflow model of computation but this model is undecidable and cannot be subject to static analysis. In contrast, the PCSDF model used by DSP Designer enables analysis of deadlock-free execution, and memory and throughput requirements. Also, CAL is a textual specification language, whereas DSP Designer provides an intuitive graphical design environment.

In summary, DSP Designer is an attempt to integrate the respective strengths of the previously discussed tools into a unified framework for hardware implementation. 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 easy integration of IPs from native and third-party libraries, like the Xilinx CoreGen library [21], which are essential to practical efficient hardware design.

III. Model Specification and Analysis

The foundation of DSP Designer is its models of computation – SDF, CSDF, and their parameterized extensions. We discuss the relevant characteristics of these models, and illustrate their suitability for specifying signal processing applications.

A. SDF and CSDF

A dataflow model consists of a set of actors inter-connected via channels. The actors represent computational units while the channels denote communication. The data is abstracted as tokens. 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. The channels store the tokens until an actor consumes the tokens.

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, then the actor may fire in an overlapping (pipelined) fashion.

Fig. 1 shows an SDF model for computing the standard deviation of non-overlapping blocks of 100 input samples each. Every actor in this model except `Sum` is single-rate or homogeneous, i.e. it consumes 1 token on every input, and produces 1 token on every output. The `Sum` actor consumes 100 input tokens and produces their sum as a single output token. Execution times of the actors vary with their implementations. `Square`, `Decrement`, `Subtract` execute in single cycle; `Divide`, `Square Root` take multiple cycles, and could be pipelined to have initiation interval of 1 cycle. `Sum` actor has execution time and initiation interval of 100.

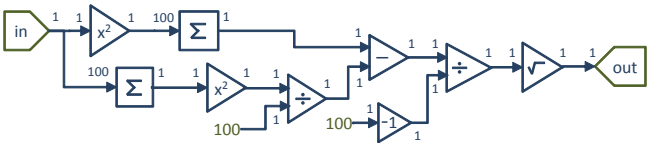

Fig. 1. Computing standard Deviations of input blocks of fixed size 100.

The SDF model of computation fits well with fixed-length computations. Such computations are abundant in signal processing standards, for example, the processing of 8×8 blocks of pixels during JPEG encoding. However, there are also computations that follow a fixed cyclic pattern in the number

of tokens processed. An example is the *normal CP* mode of LTE OFDM standard in which every slot has 7 symbols, with the first special symbol different in length from the other symbols. For such computations, the *Cyclo-Static Dataflow* (CSDF) 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 [3]. Each firing of a CSDF actor corresponds to a phase of the cyclic pattern. In Fig. 1, if we replace the input token count of `Sum` actor by a cyclic pattern $(100, 200, 300)$, then we get a CSDF model that computes standard deviation of input blocks whose lengths vary deterministically from 100 to 200 to 300 and back.

The SDF and CSDF models of computation permits efficient static analysis of key properties. The absence of deadlocks (i.e., the ability of each actor to fire infinitely often), and the consistency of execution rates (i.e., the ability to execute infinitely with bounded channels) can be verified efficiently [1] [2] [3]. Further, there are efficient algorithms for computation of throughput and buffer sizes [8], [9].

B. Parameterized Extensions

SDF and CSDF are static in nature. However, for many applications, the number of tokens processed needs to vary at run-time. For example, MP3 audio compression selects at run-time between long blocks of 576 samples and short blocks of 192 samples. Fig. 2 shows a variation of the model in Fig. 1, in which the `Sum` actors consume N tokens in each firing, where N is from the set $\{100, 200, 300\}$. This computes standard deviations of a mix of input blocks of lengths 100, 200 or 300 by varying N at run-time. This model of computation is called *Parameterized Static Dataflow* (PSDF) [4].

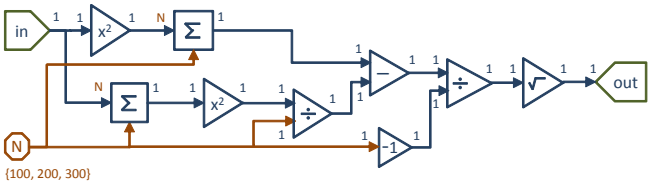

Fig. 2. Computing standard deviations of input blocks of varying size N.

The behavior of the PSDF model can be viewed as a composition of several SDF models, one for each possible value of the parameter (also referred to as configuration). Fig. 2 has 3 possible values of the parameter, hence 3 configurations. At any point in execution, the behavior of the PSDF model is the SDF model corresponding to the value of the parameter. To avoid non-determinism, a change in parameter value can take effect only at iteration boundaries. The analysis of a PSDF model accounts for the analysis for all possible configurations [4]. The CSDF model can similarly be parameterized to form the *Parameterized Cyclo-Static Dataflow* (PCSDF) model.

IV. Realizing Models in DSP Designer

DSP Designer is a graphical environment backed by the design and implementation flow of Fig. 3. In this section we describe how the user specifies applications using models, explores optimizations, and generates FPGA designs.

A. Design Flow

The user works in a graphical environment as shown in Fig. 4. 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 *Model Specification* step. The actor library consists of a rich set of primitive actors (add, square root, sine, etc.), stream manipulation actors (upsample, build stream, etc.), third-party actors (e.g. FFT and FIR blocks from Xilinx Coregen [21]), and user-defined actors that are either specified in the LabVIEW programming language or constructed using DSP Designer. This reuse of actors allows for hierarchical composition of designs within the tool.

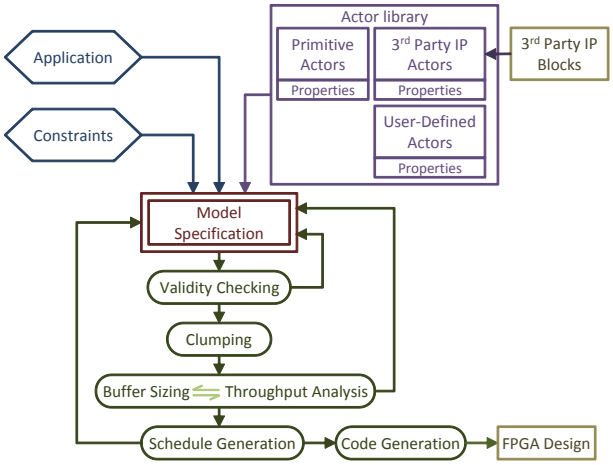

Fig. 3. Design and Implementation Flow in DSP Designer.

The user continues by connecting the actors, and optionally configuring their properties. Configurable properties of an actor include the data types and the number of tokens for its input and output channels. The number of tokens may vary at run-time for parameterized actors, depending on the current parameter value, resulting in a potentially distinct configuration for each parameter value. To ensure analyzability, the tool limits the value of each parameter to a finite set specified by the user. Some actors can also be configured for their throughput, pipeline depth, resource usage, or other implementation-specific options. The actor library includes cycle-accurate characteristics for each actor configuration, including the initiation interval and the execution time.

The second input from the user is the *Constraints*, which include minimum throughput requirements on input/output ports or internal channels of the design. Throughput is specified in engineering units, such as Mega-Samples per second (MSps).

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* includes model consistency and deadlock checking. 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 & Warning tab in the tool. On a valid design, the

tool performs *Clumping* to identify regions that fit specialized implementation schemes. *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 for the design, given the determined buffer sizes and clumped regions. This schedule is viewable in the schedule view part of the tool (as shown at the bottom of Fig. 4), providing 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.

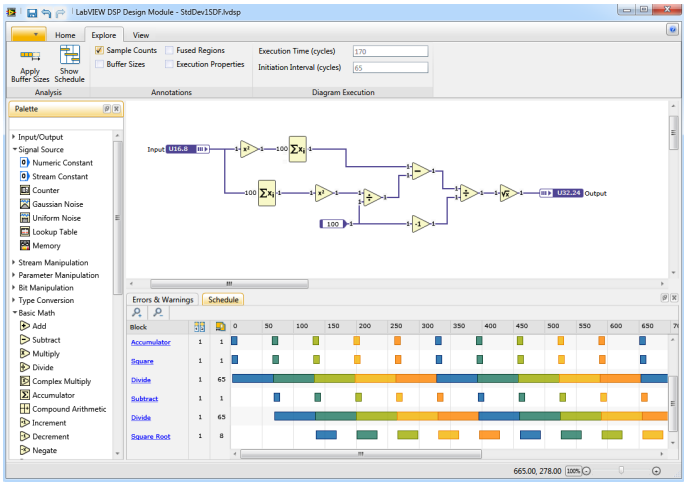

Fig. 4. DSP Designer Tool Implementing the Example in Fig. 1.

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.

B. Implementation Strategy

DSP Designer uses a FIFO-based, self-timed implementation strategy to realize the designs on FPGA fabrics [8]. 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 execution strategy every actor instance 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 on every clock cycle, allowing for a potentially more opportunistic execution than the conservative block-based model used in most SDF-based tools, where a downstream actor is not fired until a cycle after the one where its upstream actors write the last output token into their shared buffers. As a consequence, there is no global scheduling logic in this implementation strategy, reducing the complexity of the controller for each actor in the final design.

C. Actor and IP Stitching

The FIFO-based, self-timed implementation strategy is implemented using harness logic that surrounds every actor instance, providing a FIFO-based interface that realizes the SDF model and its extensions discussed in Section III. The generated code for all actors presents a standardized interface to the harnesses, based on designated lines for data and handshaking. This simplifies actor stitching since the tool can use generic harness wrapper templates. It also allows the tool to connect actors more directly and efficiently.

A faithful realization of the SDF model of computation requires extra resources for the harness logic and the FIFOs on each channel. 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 on the design to reduce both 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. The clumping activity is akin to the process of converting an asynchronous design, where all actors are connected by FIFOs, into a *GALS* [22] (Globally Asynchronous Locally Synchronous) architecture, where FIFOs connect regions of synchronously connected actors called clumps.

V. OFDM Transmitter & 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 DSP Designer.

A. System Specifications & Hardware Architecture

Our single antenna OFDM link design is based upon the LTE standard [15] 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 the National Instruments (NI) PXI Express platform shown in Fig. 6, where the transmitter (TX) and receiver (RX) consist of the following four main components.

- **PXIe-8133** Real-time (RT) controller equipped with a 1.73 GHz quad-core Intel Core i7-820 processor and 8 GB of dual-channel 1333 MHz DDR3 RAM.

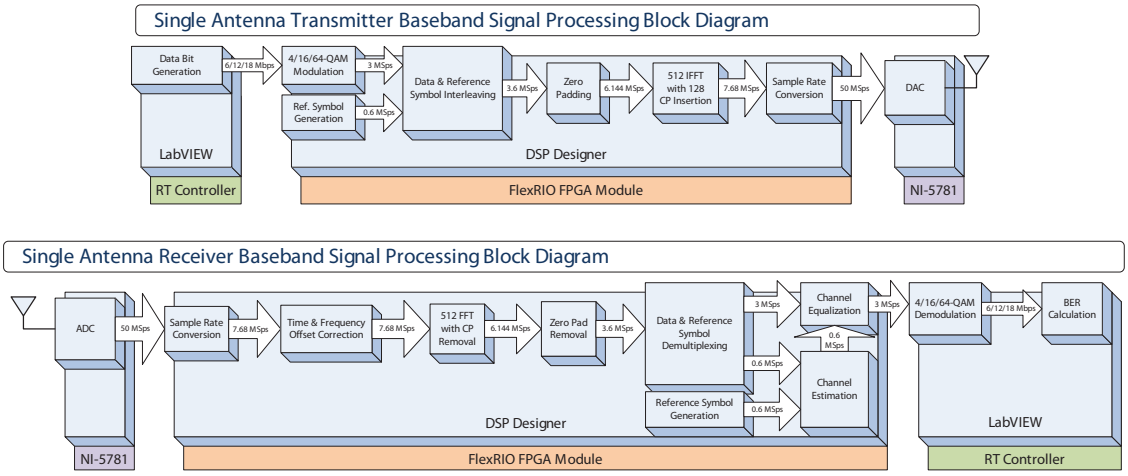

Fig. 5. Hardware and Software Mapping of Transmitter and Receiver Block Diagrams.

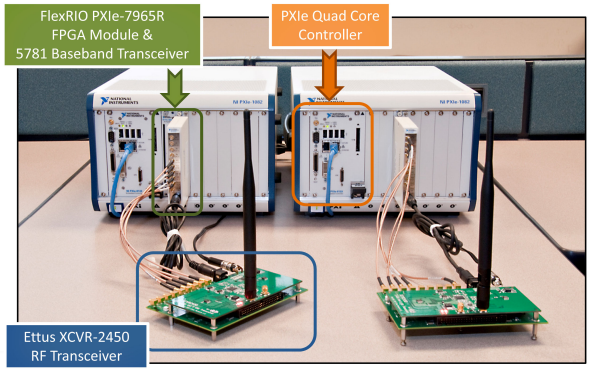

Fig. 6. National Instruments PXI Express Real-Time Signal Processing Platform with Ettus Research RF Front-End.

- **PXIe-7965R** FPGA module equipped with a Virtex-5 SX95T FPGA optimized for digital signal processing, 512 MB of onboard RAM, and 16 DMA channels for high-speed data streaming at more than 800 MBps.
- **NI-5781** 40 MHz baseband transceiver module equipped with dual 100 MSps 14-bit inputs, dual 100 MSps 16-bit outputs, and eight general purpose IO lines.
- **Ettus Research XCVR-2450** 802.11a/b/g compliant, 40 MHz, dual 2.4 GHz and 5.2 GHz band RF transceiver with 100 mW of transmit power.

Fig. 5 shows the TX and RX block diagram representations of the various signal processing blocks implemented in the devices. Also shown is a mapping of the various blocks to the underlying hardware targets and the respective design tools used in their implementation; e.g., the TX *Data Bit Generation* block (programmed using LabVIEW RT) executes on the PXIe-8133 RT controller, while the higher rate *512 IFFT with 128 CP Insertion* block (implemented using DSP Designer) executes on the PXIe-7965R FPGA module. The various data rates associated with the inputs and outputs of each block are also shown; e.g., the TX *Sample Rate Conversion* block up-samples input data streaming at 7.68 MSps to 50 MSps in order to meet the sample rate constraints of the NI-5781 DAC.

B. OFDM Transmitter Design Overview

Fig. 7 shows the DSP Designer implementation of the proposed transmitter. Random bytes of data generated by the RT controller are forwarded to the FPGA module for Multilevel QAM (M-QAM) [23]. Depending upon the modulation order value denoted by the parameterization port, Modulation, the bytes of data are unpacked into groups of 2, 4, or 6 bits corresponding to 4/16/64-QAM, respectively. Groups of bits are then mapped to their respective complex symbols and passed out of the output port of the sub-diagram.

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. Sample rate up-conversion is then performed through two sets of FIR filters, converting the 7.68 MSps signal to 50 MSps. The samples are forwarded to the NI-5781 for digital-to-analog conversion followed by RF up-conversion.

C. OFDM Receiver Design Overview

Fig. 7 shows the DSP Designer implementation of the receiver. The RX begins with two FIR filters that perform sample rate down-conversion taking the incoming 50 MSps signal from the ADC down to 7.68 MSps. Time and carrier frequency offset (CFO) estimation is performed using the blind estimation technique proposed in [24]. Because the first L samples (CP) and the last L samples of an $N+L$ length OFDM symbol are equal, the algorithm correlates the two, thereby eliminating the need for a priori knowledge of the transmitted signal. The correlation output is then used to estimate the start index of an OFDM symbol and the CFO thereof.

In order to meet throughput without loss of data during the estimation of the start index and CFO, the receive signal is buffered into a memory block while simultaneously being

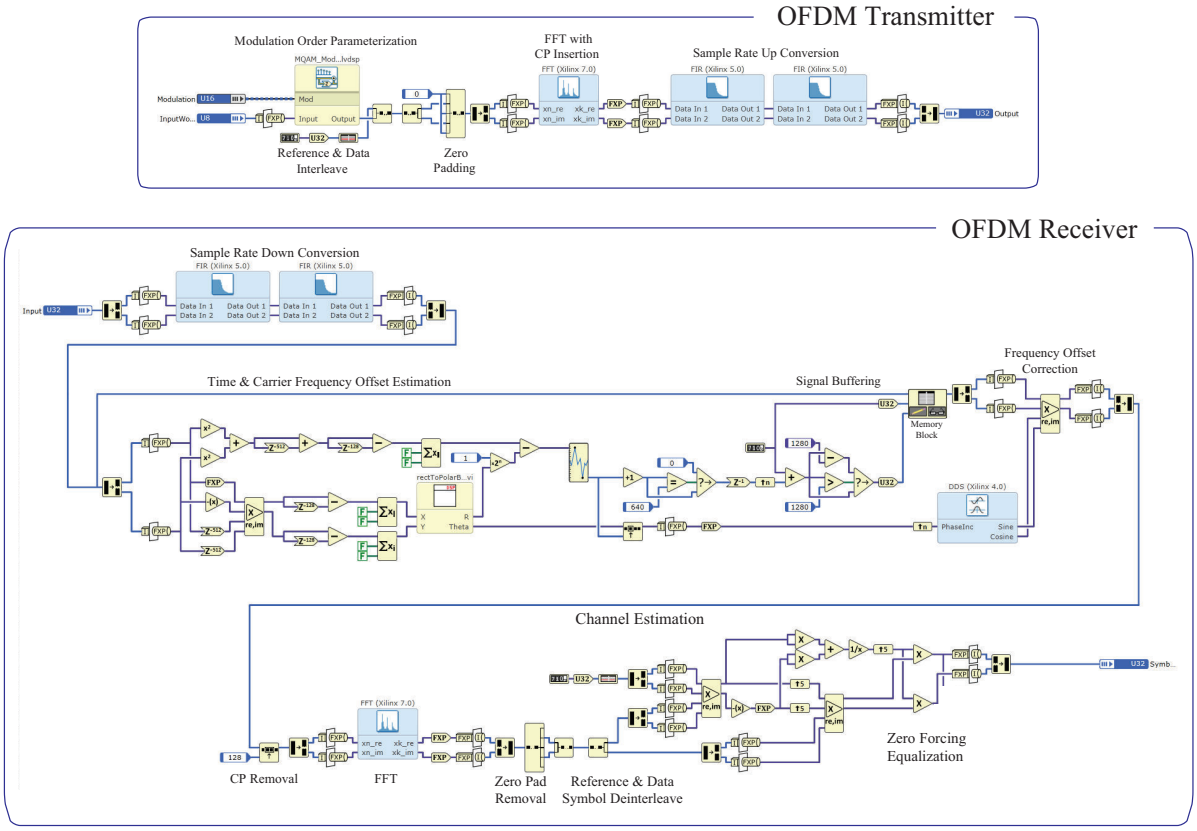

Fig. 7. DSP Designer Diagrams of OFDM Transmitter (top) and Receiver (bottom).

processed for time and CFO estimation. Once computed, the start index is used to calculate a corresponding read address pointer that indexes the beginning of an OFDM symbol stored in memory. When it is synchronized to the beginning of an OFDM symbol, the RX streams the signal out of memory and into the CFO correction block wherein the synchronized OFDM symbol is multiplied with a complex sinusoid generated by a direct digital synthesizer (DDS) block at a frequency defined by the CFO estimate present at its input.

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 deinterleave operation. As shown in Fig. 7, the received reference symbols are passed out of the first output of the deinterleave block for channel estimation while the received data symbols are passed out of the second for channel equalization.

In order to estimate the channel coefficients, we model the received reference symbols as $s_k = h_k r_k + z_k$ where $k \in \{0, \ldots, 49\}$. The reference symbol and channel coefficients are respectively modeled as $r_k = e^{j\theta_{r_k}}$ and $h_k = |h_k| \, e^{j\theta_{h_k}}$. Lastly, z_k represents additive white Gaussian noise.

The estimates of the channel coefficients, $\hat{h}_k$, are then calculated by multiplying the complex conjugate of the reference symbols, r_k^*, to the received reference symbol. Moreover, because only one reference symbol is allocated to every five data symbols, the 50 channel estimates, $\hat{h}_k$ for $k \in \{0, \ldots, 49\}$, are up-sampled by five generating a total of 250 channel estimates, $\hat{h}_i$ for $i \in \{0, \ldots, 249\}$.

In order to correct the effects of the wireless channel, zero forcing (ZF) channel equalization is employed where the received data symbols, y_i, are first multiplied by the complex conjugate of the channel estimates, $\hat{h}_i^*$, and then divided by their square magnitude, $|\hat{h}_i|^2$, effectively inverting the channel. The data symbol estimates, $\hat{x}_i$, are then transferred to the RT controller at a data rate of 3 MSps for QAM demodulation and bit error rate calculation.

D. FPGA Compilation & Run-Time Results

In addition to the portions of the design implemented in DSP Designer, 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 RT controller. The results also include additional logic to control the NI-5781, such as ADC/DAC read/write operations, sampling frequency configuration, and clock select.

Table I is a 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 of the various resources for the TX and RX are listed in the last two columns. For instance, there are 14,720 slice elements available on each FPGA, 43.1% or 6,350 of which are used by the TX and 79.2% or 11,659 of

which are used by the RX. Due to significant differences in computational complexity between the two designs, the RX utilizes more than twice as many slice registers and LUT resources compared to the TX. With regard to timing, the TX and RX DSP diagrams are configured to be driven by 125 MHz clocks, and both successfully met timing during compilation.

Resource Name	Available Resource Elements	Transmitter Utilization	Receiver Utilization
Slices	14,720	43.1%	79.2%
Slice Registers	58,880	21.6%	54.6%
Slice LUTs	58,880	24.7%	57.3%
DSP48s	640	2.7%	8.3%
Block RAM	244	8.2%	19.7%

TABLE I
FPGA RESOURCE UTILIZATION SUMMARY.

Fig. 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 $8 * 10^{-4}$.

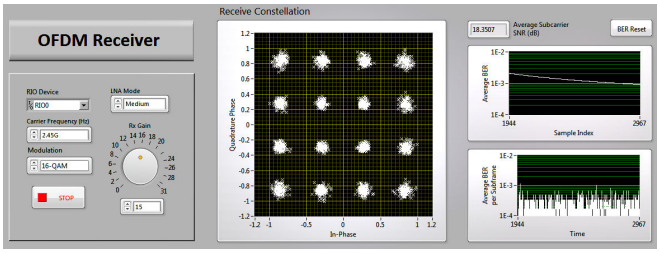

Fig. 8. Receiver Front Panel.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presented the DSP Designer framework to specify dataflow models, analyze them, and generate implementations for hardware targets. The PCSDF model of computation is sufficiently expressive in specifying complex streaming applications, while capturing characteristics specific to hardware design. The back-end performs key optimizations related to buffer sizing and scheduling. The actor library provides a rich set of building blocks to create complex signal processing and communications applications. It also facilitates easy integration of custom designed hardware IPs from native and third-party sources. Thus, DSP Designer serves as a design and exploration framework that enables algorithm experts to productively specify applications using high level models and still create efficient hardware implementations.

In the future we intend to (a) extend the DSP Designer modeling and analysis capabilities to support more expressive streaming models such as Heterochronous Dataflow (HDF); (b) enhance the analysis back-end to address problems related to dataflow pipelining and resource constrained scheduling; (c) study how intra-cycle timing optimizations for hardware, such as retiming and recycling, can be applied at the model level; (d) derive more resource-efficient hardware implementations through rate matching and clumping of multi-rate actors; and (e) enlarge the DSP Designer actor library and standardize the IP interface definition to ease third-party IP integration.

REFERENCES

[1] E. A. Lee and D. G. Messerschmitt, "Synchronous Data Flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sept. 1987.

[2] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs*. Norwell, MA: Kluwer Academic Press, 1996.

[3] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cyclo-static data flow," in *IEEE Intl. Conf. Acoustics, Speech, and Signal Processing*, vol. 5, 1995, pp. 3255–3258.

[4] B. Bhattacharya and S. Bhattacharyya, "Parameterized Dataflow Modeling for DSP Systems," *Signal Processing, IEEE Transactions on*, vol. 49, no. 10, pp. 2408 –2421, oct 2001.

[5] A. Girault, B. Lee, and E. A. Lee, "Hierarchical Finite State Machines with Multiple Concurrency Models," *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 6, pp. 742–760, June 1999.

[6] B. D. Theelen, M. C. W. Geilen, T. Basten, J. P. M. Voeten, S. V. Gheorghita, and S. Stuijk, "A Scenario-aware Data Flow Model for Combined Long-run Average and Worst-case Performance Analysis," in *Proceedings of MEMOCODE'06*, Jul. 2006, pp. 185–194.

[7] S. Tripakis, H. Andrade, A. Ghosal, R. Limaye, K. Ravindran, G. Wang, G. Yang, J. Kormerup, and I. Wong, "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*, ser. CODES+ISSS '11. New York, NY, USA: ACM, 2011, pp. 59–68.

[8] O. M. Moreira and M. J. G. Bekooij, "Self-Timed Scheduling Analysis for Real-Time Applications," *EURASIP Journal on Advances in Signal Processing*, vol. 2007, no. 83710, pp. 1–15, April 2007.

[9] S. Stuijk, M. Geilen, and T. Basten, "Exploring Trade-offs in Buffer Requirements and Throughput Constraints for Synchronous Dataflow Graphs," in *Proceedings of DAC '06*, 2006, pp. 899–904.

[10] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming Heterogeneity - The Ptolemy Approach," in *Proc. of the IEEE*, vol. 91, no. 1, 2003, pp. 127–144.

[11] H. A. Andrade and S. Kovner, "Software Synthesis from Dataflow Models for G and LabVIEW," in *In Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers*, 1998, pp. 1705–1709.

[12] The MathWorks Inc., "Simulink User's Guide," 2005, http://www.mathworks.com.

[13] H. Kee, C.-C. Shen, S. Bhattacharyya, I. Wong, Y. Rao, and J. Kornerup, "Mapping Parameterized Cyclo-static Dataflow Graphs onto Configurable Hardware," *Journal of Signal Processing Systems*, pp. 1–17, 2011.

[14] H. Berg, C. Brunelli, and U. Lücking, "Analyzing Models of Computation for Software Defined Radio Applications," in *International Symposium on System-on-Chip (SOC)*, Tampere, Finland, November 2008, pp. 1–4.

[15] "3GPP LTE: The Mobile Broadband Standard," Dec 2008, http://www.3gpp.org/.

[16] National Instruments Corp., "LabVIEW FPGA," www.ni.com/fpga.

[17] Xilinx Inc., *System Generator for DSP: Getting Started Guide*, www.xilinx.com.

[18] C.-J. Hsu, J. L. Pino, and F.-J. Hu, "A mixed-mode vector-based dataflow approach for modeling and simulating lte physical layer," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: ACM, 2010, pp. 18–23.

[19] J. W. Janneck, "Open Dataflow (OpenDF)," http://www.opendf.org/.

[20] J. Janneck, I. Miller, D. Parlour, G. Roquier, M. Wipliez, and M. Raulet, "Synthesizing Hardware from Dataflow Programs: An MPEG-4 Simple Profile Decoder Case Study," in *IEEE Workshop on Signal Processing Systems*, oct. 2008, pp. 287–292.

[21] Xilinx Inc., *Xilinx Core Generator, ISE 12.1*, Xilinx Inc., 2010.

[22] D. M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems," Ph.D. dissertation, Stanford Univ., CA., October 1984.

[23] J. Proakis, *Digital Communications*, 4th ed. McGraw-Hill Science/Engineering/Math, Aug 2000.

[24] M. Sandell, J.-J. van de Beek, and P. O. Brjesson, "Timing and Frequency Synchronization in OFDM Systems Using the Cyclic Prefix," in *In Proc. Int. Symp. Synchronization*, 1995, pp. 16–19.