Increasing Design Productivity Through Core Reuse, Meta-Data Encapsulation, and Synthesis

Adam Arnesen, Kevin Ellsworth, Derrick Gibelyou, Travis Haroldsen, Jared Havican, Marc Padilla,
Brent Nelson, Michael Rice, and Michael Wirthlin
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Dept. of Electrical and Computer Engineering
Brigham Young University
Provo, UT, 84602, USA

***Abstract*—This paper presents a novel IP core reuse strategy which reduces design time from days to hours for communication circuits such as digital radio receivers. This design productivity is obtained by leveraging a highly parameterized library of communication specific cores. These cores are described in IP-XACT XML with vendor extensions describing the timing behavior of their communication interfaces. A synthesis tool, called Ogre, was created that generates the communication interfaces between cores described in IP-XACT and synthesizes full designs from structural synchronous dataflow specifications. Design productivity improvements are demonstrated with several radio receiver designs.**

I. Introduction

Reusing hardware cores has long been held out as a promising approach for improving design productivity [1]. Indeed, it has successfully been used in System-on-chip (SOC) design flows [2], where cores are integrated together using well defined bus interfaces such as AMBA, PLB (core connect), and Wishbone. As long as the cores correctly adhere to a given bus protocol, hardware reuse involves simply the structural interconnection of a set of predefined cores which have been parameterized (i.e., address range, interrupt behavior, etc.) Meta-data formats such as IP-XACT [3] have further been proposed for encapsulating many of the low-level details of a reusable core in machine readable form. Finally, design tools have been created to use this meta-data such as Xilinx EDK, Altera SOC Builder, and Mentor Graphics Platform Express.

In spite of these successes, reuse is much more difficult in general hardware design scenarios where non-standardized interconnect and communication protocols are used. Examples of non-standard communication include point to point connections, broadcast connections, and various types of handshaking and timed exchanges of data. Often, these protocols are custom to each module and thus require the creation of additional circuitry to interconnect a collection of cores to one another. Supporting the automatic creation of this interface circuitry requires that the timing and protocol information of these modules be characterized in a machine readable manner. In addition, a corresponding CAD tool is required with the ability to read this timing information and use it to correctly and automatically interconnect a collection of cores into a fully scheduled, finished design.

This work was supported by the I/UCRC Program of the National Science Foundation under Grant No. 0801876.

This paper describes such an interface synthesis approach and associated CAD tool set (called Ogre). While it does not address the automated reuse of circuit modules with *arbitrary* interconnection protocols, it moves beyond bus-based interconnection approaches and introduces timing parameterization for coarse grain hardware modules in stream-based processing applications which implement the synchronous data flow (SDF) model of computation. It does this by introducing a set of vendor extensions to the recently-approved IP-XACT standard and which describe the timing behavior of cores in a streaming computation. Specifically, these extensions describe point to point connections between producing and consuming SDF actors that are defined by their consumption rates, latency, and sample delay. In addition, they provide for the additional parameterization of cores at high levels of abstraction, simplifying the use of those modules by designers. The corresponding CAD tool flow processes these IP-XACT descriptions of module timing behavior and synthesizes user designs into efficient implementations.

To demonstrate the effectiveness of the technique, a digital radio building block set was created and described using these IP-XACT extensions. The Ogre CAD tool was then used to rapidly implement a variety of digital radios including seven different QPSK implementations. The results demonstrate that the Ogre design flow reduced design time for the selected radios from a few days to less than an hour. Seven different QPSK designs, each of which occupy a different location in the area/time tradeoff space, demonstrate the ability of Ogre to support rapid design space exploration and find a variety of solutions to a problem instance by automatically handling many of the timing details for the user.

The balance of the paper is as follows. First, our approach to the parameterization of circuit modules is introduced along with an introduction to IP-XACT and our extensions. The parameterized block library which was used for the experiments in this paper is introduced. The Ogre CAD tool flow is then described followed by a description of a set of experiments designed to demonstrate the utility and performance of the

Parameter	Type	Description
loopBandwidth	double, 0.01 to 0.5	B_nT used in calculating constant multiplicand values
loopDampingFactor	double, 0.5 to 1.5	ζ used in calculating constant multiplicand values
phaseDetectorGain	double, -10.0 to +10.0	K_p used in calculating constant multiplicand values
accumulationWidth	natural	Number of bits right of radix point for internal accumulator
kPrecision	positive ($\leq$ 62 bits)	Number of fractional bits used for constant multiplicand values
ddsGain	double, -10.0 to +10.0	K_0 used in calculating constant multiplicand values
samplesPerSymbol	$\{2, 4\}$	N
order	$\{1, 2\}$	Loop order: first order (no accumulator) or second order (with accumulation)

TABLE I
PARAMETERS FOR LOOP FILTER CORE.

approach. Finally, conclusions and suggestions for future work are given.

II. Library Core Parameterization and Meta-Data Representation

Comprehensive parameterization of cores significantly increases their reusability. Cores that have many parameters can be used in a variety of situations with no changes to the core. Although it is more difficult to create highly parameterized cores—the extra design time is offset by the increased design productivity provided by this parameterization.

While a highly parameterized library of cores helps to enable reuse and increases design productivity, significant increases in productivity are further enabled through meta-data descriptions of these IP cores. Meta-data in electronic formats enables *automated tools* to reason about details of core operation and interconnect and abstracts these details from the designer. This work represents this meta-data in XML by leveraging the standard IP-XACT XML schema for representing IP cores [3], [4]. The strengths of IP-XACT for describing large libraries of cores for SOC are discussed in detail in [5] and include strong taxonomy and core naming, hardware port information, file sets, and bus-based interconnection schemes. This work leverages IP-XACT to represent parameters and extends it to describe the timing of core interfaces. Specifically, this work represents the following four types of parameters in a meta-data format:

Low-Level Parameterization: This is most closely related to conventional VHDL-level parameterization. That is, it is used to declare bit-widths on inputs and outputs, rounding modes, pipelining directives, etc.

Domain-Specific Parameterization: This is used to provide a level of parameterization usable by a domain expert who is not necessarily a hardware designer. It allows a domain expert to deal with cores at the levels of abstraction he or she typically deals with when designing application-specific models.

Dependent Parameterization: This describes the link between domain-specific and low-level parameterization. It allows low-level parameters to be computed based on domain-specific parameter values set by an application expert.

Temporal Parameterization: This refers to the parameterization introduced in this work to describe the temporal behavior of a core.

It is important to understand our use of the word "parameterization" here. Often, this refers to the parameters passed to a module generator program and which controls the creation/generation of the module instance. However, our use of the term parameterization here refers more to a description of the behavior of an already-existent core without regard to how it was generated. That is, our goal is to describe existing cores at a level of detail which will enable a design tool to combine a collection of cores together with custom-synthesized interface circuitry to create a finished design. In this section we discuss these types of parameterization and how they were implemented using the IP-XACT standard.

A. Low-Level Parameterization

Parameterization of hardware cores has traditionally been done with low-level parameters such as bit-widths and operating modes. This level of parameterization increases the reusability of a core by allowing it to be used in designs that require different bit-widths. Low-level parameterization is quite common and is fairly simple to implement, especially when the cores are intended to be generated using module generators. The cores developed in this research are parameterize at this low-level.

Low-level parameterization (as with all the parameterization done in this work) leverages the standard IP-XACT XML schema for representing IP cores [3], [4]. Parameterization is a native feature of IP-XACT and allows parameters to be given default values, provides a manipulation interface for these parameters and allows parameters to be set by the user, by a tool, or to be set based on other parameter values.

B. High-Level Parameterization

To further increase the reusability of cores, additional high-level parameters specific to digital communication systems are used. These high-level parameters allow the designer to interact with the core at a higher level of abstraction. Examples of high-level parameters are shown in the Table I for a loop filter core. This is a simple first-order loop filter consisting of a multiplier and accumulator. In addition to low-level parameters for signal bit-widths and constant multiplication coefficients, this core contains high-level parameters specific to communication receivers. The parameters shown in the table are used to perform a non-trivial calculation which determines both bit-widths and coefficients necessary for a given signal processing function. In addition, this core contains a parameter named 'samplesPerSymbol'—this allows the core to be quickly used in different radio personalities that each use a different number of samples for each output symbol computed. Changing this parameter fundamentally changes the core's internal organization. This high level of parameterization significantly improves the ease of reuse for these cores.

C. Dependent Parameterization

Once high-level parameters have been set, many low-level parameters can be automatically computed by evaluating expressions found within the IP-XACT description of the core. These expressions leverage the XPath expression language provided natively by XML [6] and custom expression extensions and functions developed within this work [4]. This reduces the number of unique parameters that must be specified yet allows for powerful, deep parameterization. We note here that not all of the dependencies between high and low-level parameters are expressed in IP-XACT—VHDL functions (in corresponding core module generators) are also leveraged for the computation of some low-level parameters.

The XML example below defines a dependent parameter that is evaluated based on a high-level parameter named 'Sregsize'.

```
<spirit:value spirit:resolve="dependent"
    spirit:dependency="(id('Sregsize') &gt;= 2)
    * id('Sregsize') + (id('Sregsize') &lt; 2)
    * 2">2
</spirit:value>
```

D. Temporal Parameterization

In order to facilitate the automatic interconnection of cores, the temporal behavior of the core interfaces needs to be expressed. While IP-XACT provides most of the elements needed to describe cores, some additional information is needed to enable CAD tools to reason about the timing of cores and their interconnect in data-driven applications. IP-XACT allows for external vendor extensions to support this extra information. This research extends IP-XACT in several ways including extensions to describe the temporal behavior of library cores.

As described earlier, the design space is constrained to designs modeled by the homogeneous synchronous data-flow model of computation [7]. Meta-data descriptions must be added to IP-XACT to describe the temporal behavior of this model. The following three elements are added to IP-XACT via vendor extensions to define this behavior: latency, data introduction interval and sample delay. Each of these extensions are represented by XML elements as extensions to the IP-XACT schema.

The latency represents the number of clock cycles that elapse from the time that data is consumed on the inputs of the core to the time that the corresponding results are produced on the outputs. This does not mean that the core is pipelined in the traditional sense or that data can be accepted by the core on every cycle. For example, cores that accept data only every 8 cycles and take 9 cycles to compute a result would be given a latency value of 9. The Ogre scheduler uses this information to determine appropriate start times for pipelined cores. All cores used in this environment have a static latency to allow the scheduler to perform static scheduling.

The data introduction interval for a core describes how many clock cycles must elapse between the introduction of data for each new sample. Cores with a data introduction interval of one can accept new samples each clock cycle. The data introduction interval of a core is independent of its latency. For example a core that has a data introduction interval of 3 can consume data on clock cycle 0 but then will not consume data again until clock cycle 3 and then again on cycle 6.

The sample delay parameter indicates the number of SDF sample delays (Z^{-1}) that occur between the input of the core and its output. These are different delays than regular pipeline register delays. Sample delays are used during synthesis to eusure that samples correctly line up when pipelined cores are used for computation. The outputs of a core with a sample delay of one would be used in computations with the sample immediately following the one that produced that particular output.

The XML example below defines a temporal SDF interface with a data introduction interval of 7, a pipeline depth of 8, and a sample delay of 0.

```
<chrec:behavioralLayer>
  <chrec:dataIntroductionInterval>7
  </chrec:dataIntroductionInterval>
  <chrec:pipelineDepth> 8
  </chrec:pipelineDepth>
  <chrec:sampleDelay>0</chrec:sampleDelay>
</chrec:behavioralLayer>
```

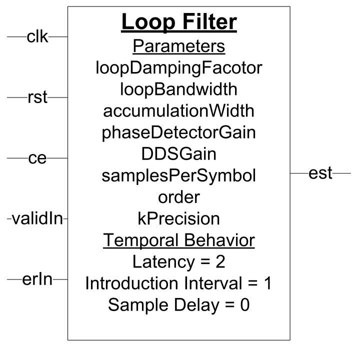

Fig. 1. This figure shows the loop filter block from the library. This block should be modeled as having a latency of 2 and a data introduction interval of 1 and no sample delay.

Figure 1 provides an example of the top-level interface of the loop filter block. This figure shows the level of parameterization that is presented to the user. Note that the low-level details of bitwidths and internal behavior are largely hidden. The temporal behavior listed is used internally by the tools and in this example show that the core takes two clock cycles to compute one result, it can accept new data every clock cycle, and there is no internal sample delay.

III. A Library of Parameterized Cores

The design of digital radio receivers was chosen as the demonstration vehicle for this work. To that end, a library of building blocks suitable for the creation of a variety of radio personalities was developed. The blocks were first created as parameterized VHDL modules. Meta data descriptions of these cores were then created in the IP-XACT format discussed earlier. The temporal characteristics of the cores were specified to facilitate their use within a SDF system.

The radio receiver personalities targeted in this research include QPSK, Offset QPSK, PCM/FM, 16QAM, 8PSK, and 16APSK, although other desired constellations may also be possible with slight adjustments. The creation of the block set took advantage of the fact that there are many recurring blocks in these different radio types. These recurring blocks include interpolators, timing error detectors (TEDs), phase error detectors (PEDs), loop filters, direct digital synthesis (DDS) blocks, and numerically controlled oscillators (NCOs). A list of some of the created blocks is as follows:

Clockwise Rotation: Rotates a complex signal by a certain angle, determined by sine and cosine inputs.

Interpolator: FIR filter which outputs an approximation of the desired sample based on available sample values.

Decision Block: Finds and outputs the constellation point for a given modulation scheme that is closest to the processed input value.

Timing Error Detector: Computes the sample timing error. The rest of the blocks in a typical timing loop attempt to drive this error to zero.

Loop Filter: A proportional-plus-integrator filter. This is commonly used to smooth the output error signals coming from the TED and PED cores.

Numerically Controlled Oscillator: Part of timing loop control; generates control signals for TED and PED.

Calculate Mu: Generates fractional interval, μ, typically for use by interpolator.

Phase Error Detector: Computes a sample phase error. The rest of the blocks in a typical phase loop attempt to drive this error to zero.

Direct Digital Synthesizer: Generates sine and cosine outputs based on an input phase value.

One of the goals of this work was to support the exploration of cost/performance points in the overall solution space for each radio personality selected. Thus, multiple versions of each block were designed which differ in their temporal behavior. For each block there are combinational versions as well as heavily pipelined versions to facilitate different radio requirements. In addition, the various blocks exhibit different data consumption rates based on their internal design. Table II lists three blocks in the library and their variations. For example, four cubic interpolator blocks are available to support a range of latencies, clock rates, and resource requirements. Not shown in the table, each cubic interpolator also has a different data consumption rate.

Block	Latency (cycles)	Delay (ns)	Max Freq. (MHz)	Area	
				Slices	DSPs
Cubic Interp-olator	0	34.5	N/A	22	12
	8	43.1	185	53	12
	9	54.6	164	156	4
	16	43.8	365	119	1
TED	0	9.6	N/A	53	2
	1	12.5	159	53	2
	2	10.5	284	55	2
Loop Filter	0	11.1	N/A	66	5
	2	18.0	167	74	5
	3	18.0	223	74	5

TABLE II
A listing of different versions of blocks that were created and their timing/area characteristics. Latency is measured in clock cycles and is therefore omitted in combinational versions which have no input clock. Block Delay is the total time from when the input is presented to when the output corresponding appears.(These results are based on a Virtex4-SX35 FPGA)

IV. The Design Environment: Using Cores and Their Descriptions

A rich library of parameterized cores and their XML meta-descriptions are not useful unless a design tool is available that can interpret the XML. Such a design tool should be able to import the XML-based cores and make them available to the designer. Because the tool can interpret the parameters of the cores, the tool can make user resolved parameters available to the designer and automatically set others to computed values as defined by XML dependencies. A variety of different design tools could be created at different levels of abstraction to integrate these cores such as high-level compiler tools, structural design tools, or application-specific design tools.

A structural design tool was created in this project that allows a designer to compose digital radio receivers (and other signal processing systems) by selecting circuit cores from a library and defining the communication between them. The structural design is performed using the well-known Simulink GUI. HDL cores, with their associated XML wrappers, are imported into a Simulink library and placed on a GUI panel. The communication between the cores is defined using Simulink connections. The designer also sets the high-level parameters available for each core (low level parameters are set by the design tool). An example of a radio model created within this GUI is seen in Figure **??**.

The design tool synthesizes a netlist through the following steps as shown in Figure 2:

1) Checks the structural integrity of the design and validates the parameters set by the user.

2) Propagates high-level parameters set by the user to the low-level parameters within the design.
3) Resolves bit-widths and sets the bit-widths of all unconstrained ports and signals.
4) Generates a global schedule for the design determining the optimal start times of all cores using an iterative modulo scheduling technique [8].
5) Synthesizes a finite state machine controller in VHDL to sequence the cores according to the generated schedule.
6) Generates the top-level HDL model in VHDL that instances and interconnects the given cores (with their instance specific parameters) and the finite state machine.

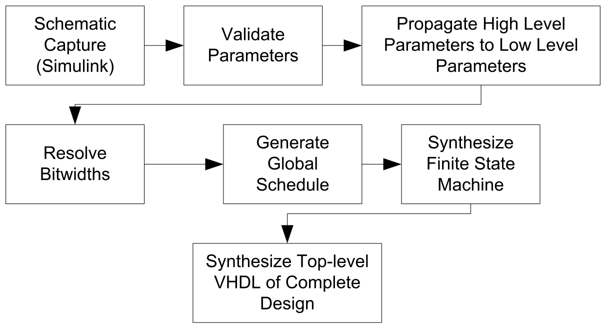
Schematic Capture (Simulink)
Validate Parameters
Propagate High Level Parameters to Low Level Parameters
Resolve Bitwidths
Generate Global Schedule
Synthesize Finite State Machine
Synthesize Top-level VHDL of Complete Design

Fig. 2. The synthesis tool flow produces synthesizable VHDL from the Simulink-based schematic capture.

After completion of the flow, the generated design and core VHDL files are ready to be used in the normal FPGA design tool flow. All of the designs generated by this research were able to be fully synthesized and placed and routed onto a Xilinx SX-35 FPGA on an XTremeDSP board. The results of these system's generation is discussed in Section V.

V. Experimental Results

To test the core library and its associated meta-data in the design environment, several radio designs were created both by hand and by using the tool.

A. Manual Constructing Radios

After completion, the library of cores was used to construct two different radios. The first was a basic combinational QPSK demodulator which consumed one data sample per clock cycle (see Figure 3).

Construction of the combinational QPSK radio was fairly straightforward. Connecting the cores by hand, it took about a day to produce a working radio that could run on hardware with a zero bit error rate. This radio test was fairly uninteresting but it proved that the VHDL cores were functionally correct.

The second design consisted of pipelined versions of many of the cores and was thus able to run at a much higher clock rate. However, the pipelined cores increased the complexity of

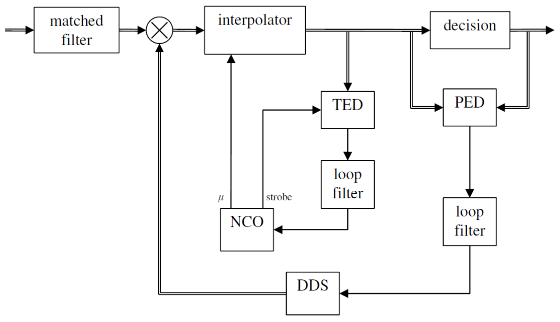

Fig. 3. A QPSK Receiver

the radio design because of the feedback and use of pipelining. After determining the desired timing and sequencing required, a finite state machine controller was created to control and sequence the blocks in the new design. Finally, a number of manual design iterations were required to find a solution in the design space which was able to meet both sample and cycle-level timing. The final design required 15 clock cycles per loop iteration (input sample) and the design time was approximately three days.

B. Automatic Radio Generation Using Ogre

The first design created using the tool flow of the previous section was the combinational QPSK receiver. The design time, which was a day for the hand-connected design, was reduced to less than an hour when using the tool. The blocks were simply interconnected in Simulink and the Ogre tool completed the design. The pipelined version of the QPSK receiver discussed above was also implemented using the Ogre design environment, producing a functional radio in less than an hour, a significant improvement over the previous three day design time. Not only was design productivity improved, but the loop schedule or latency, which was 15 clock cycles in the hand-built design, was shortened to 14 cycles without a decrease in reachable clock frequency.

Design space exploration was also facilitated by this rapid design generation. Instead of taking a few days to get one design working, it was now possible to get many designs working in a single day. This drastically changes the design process. The question changes from "How can I get this combination of cores to function correctly?" to "Which core combination is best?"

This design time improvement can also be leveraged to rapidly create an entire suite of radios with different characteristics. Using the entire reuse system, this research was able to produce seven different QPSK implementations, a BPSK design, and 8PSK and 16QAM designs in a single day. Each of these designs was implemented and downloaded to the XTremeDSP board and demonstrated to correctly produce a constellation. Bit error plots were generated for several QPSK designs as shown in Figure 4. These plots show that the generated QPSK design performed comparably to the hand-

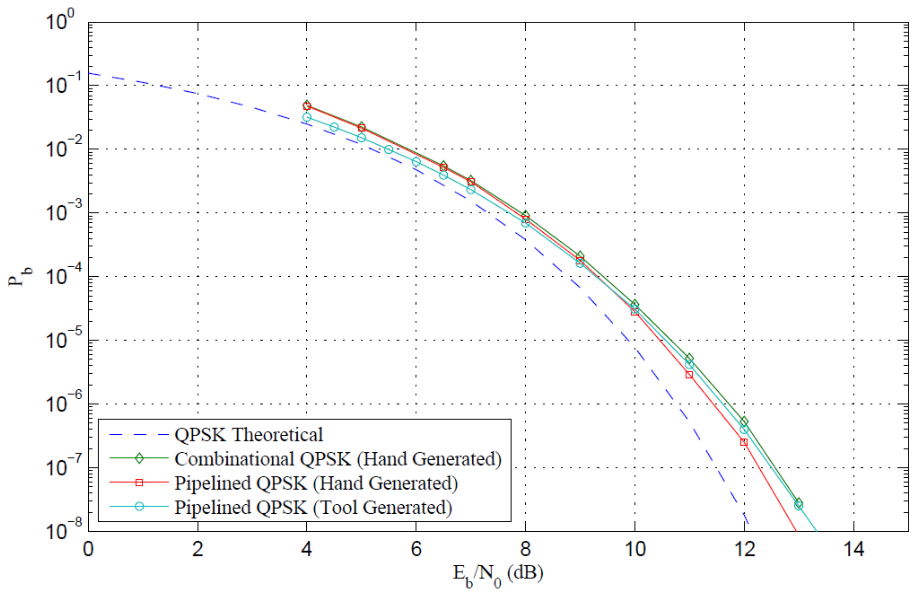

Fig. 4. Bit error rate tests were performed on the generated and hand built radios to verify correctness.

built design. No performance decrease was observed. This type of design time improvement can contribute not only to an increase in design productivity but also to the feasibility and ease of use for rapidly reconfigurable radio and other data-driven designs. For example, radio systems implemented in FPGAs could be rapidly designed and configured on the fly to meet current needs in the field.

VI. Conclusion

This paper has presented an approach for reusing hardware cores in an end to end design environment and has demonstrated significant design productivity improvements with this system. These productivity improvements have been seen previously in SOC design and this research extends these improvements to data-driven design by developing a library of standard cores for digital communication systems, describing these cores in meta-data, and leveraging the library and its core descriptions in a design environment. The library of cores developed in this research consists primarily of cores for use in digital communication systems. These cores are parameterized at both a high and low level to provide flexibility and to allow the designer to reason with the cores at a higher level. These cores also differ in their latency, data introduction interval and sample delay characteristics and each functions as an actor in an SDF graph.

Each core in the library is accompanied by a meta-data description which encapsulates all of the details of the cores' operation. This is done by leveraging and expanding the standard IP-XACT schema. Parameterization native to IP-XACT is extensively used and extensions are added to fully specify the temporal behavior of the core.

The cores and their accompanying meta-data are used in a design system which allows the designer to quickly specify a circuit's structure in a GUI. This structure is then given to the synthesis system which ensures that all cores are correctly connected to one another and that the control signals on these cores are correctly set to enforce all data dependencies and to ensure proper operation.

The combination of reusable cores, meta-data describing these cores, and an end-to-end synthesis flow enables significant improvements in design productivity. Designs that took weeks to build by hand were built in a matter of hours using the library, descriptions, and tools. These productivity benefits were observed on several different radio designs.

References

[1] *International Technology Roadmap for Semiconductors 2007 Edition: Design*, International Semiconductor Industry Association, 2007.

[2] R. Bergamaschi, S. Bhattacharya, R. Wagner, C. Fellenz, M. Muhlada, F. White, J.-M. Daveau, and W. Lee, "Automating the design of SOCs using cores," *Design & Test of Computers, IEEE*, vol. 18, no. 5, pp. 32–45, Sep-Oct 2001.

[3] *IP-XACT Draft/D5: A specification for XML meta-data and tool interfaces*, SPIRIT consortium, 1370 Trancas Street #184, Napa, CA, 94558, May 2009.

[4] *IP-XACT v1.4: A specification for XML meta-data and tool interfaces*, SPIRIT consortium, 2008.

[5] A. Arnesen, N. Rollins, and M. Wirthlin, "A multi-layered XML schema and design tool for reusing and integrating FPGA IP," in *19th International Conference on Field Programmable Logic and Applications (FPL-2009)*, August 2009, pp. 472–475.

[6] W. Wide Web Consortium (W3C), "XML Path Language (XPath) 2.0," http://www.w3.org/TR/xpath20/, January 2007.

[7] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[8] B. R. Rau, "Iterative modulo scheduling," *The International Journal of Parallel Processing*, vol. 24, no. 1, February 1996.