Meta-data and Interface Synthesis Techniques for Improving Design Productivity in Reconfigurable Computing

Adam Arnesen
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Dept. of Electrical and Computer Engineering
Brigham Young University
Provo, UT, 84602, USA
adamarnesen@byu.net

***Abstract*—This paper demonstrates improvements in design productivity for reconfigurable computing which are accomplished through a novel IP reuse strategy. It presents a set of extensions to the IP-XACT XML specification that define the temporal behavior of cores and describes how these extensions are used in the Ogre synthesis system to simplify design complexity and thereby reduce design time. Design productivity improvement is demonstrated by reducing design time for software radio designs from days to hours.**

I. Introduction: Design for Reconfigurable Computing

Reconfigurable platforms such as Field Programmable Gate Arrays (FPGAs) promise to decrease the time to market for computing systems when compared with their custom silicon ASIC counterparts. This decrease is in part a result of being able to rapidly implement designs on FPGAs because they are mapped onto pre-produced silicon and therefore do not require multiple fabrication cycles to produce. Also software development time for FPGA-based systems is also shorter as actual hardware is available for software development early in the design cycle [1].

The unique architecture of FPGAs enables designers to implement a nearly infinite number of hardware designs on the same chip without having to produce a custom silicon solution. The basic hardware layout for an FPGA is shown in Figure 1. This architecture consists fundamentally of an array of processing elements with routing resources between them. Each processing element can implement a logic function and either save its result in a memory element or directly pass it on to another processing element. To implement a design on an FPGA, a designer simply writes a hardware description language (HDL) description of the circuit he wishes to implement, the design is then synthesized into a set of logic equations and memories which are mapped into the logic elements of the FPGA. Each logic element is then connected to other elements by turning on programmable interconnection points between the wires in the mesh as shown in Figure 1. This architecture makes FPGAs especially appropriate for computations that are easily broken down into blocks that perform a part of the computation and then pass data to the next block for further work. These types of computations can typically be modeled by the synchronous data-flow (SDF) model of computation [2].

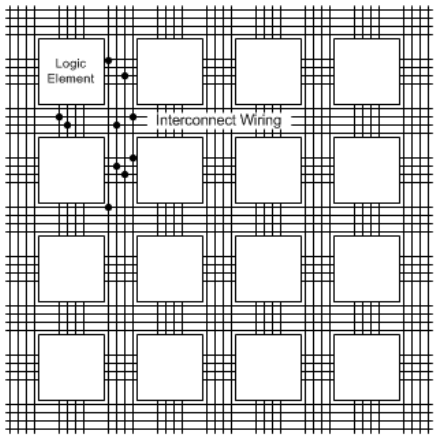

Fig. 1. A basic FPGA has logic elements containing look up tables and registers surrounded by a mesh of interconnect wires. Interconnect wires can be programatically connected to one another to rout signals between processing elements(shown by black dots in the figure).

Many algorithms that have traditionally been implemented in software on a microprocessor can be mapped to custom hardware in an FPGA. This mapping allows algorithms with inherrent data parallelism to operate much more rapidly by operating in "custom" hardware. While FPGAs have shortened the time-to-market and allowed large computational speedups for many applications, low design productivity has been a significant barrier to their widespread adoption. *Reuse* of previously designed and verified intellectual property (IP) cores in an important method for improving design productivity; however, reuse has proven difficult because it requires a

This work was supported by the I/UCRC Program of the National Science Foundation under Grant No. 0801876. and by the Rocky Mountain NASA Space Grant Consortium

system designer to understand the internal details and interface protocol of a particular core and to build custom circuitry to integrate this core into a design. If a designer can produce this same core themselves from scratch in 30% of the time it takes to reuse the core, reuse will fail [3].

The reuse problem has been addressed in part by describing IP cores in computer-readable meta-data. The IP-XACT XML schema from the Spirit Consortium defines a standard way of describing reusable cores [4]. This specification primarily describes cores that have standard interconnection schemes and can be attached to standard bus structures for communication. IP-XACT does not, however, provide meta-data for describing custom interfaces that do not comply to a standard protocol.

This paper presents a novel reuse framework that includes a highly parameterized core library for software radio, extensions to the IP-XACT schema allowing the library to describe cores whose interfaces do not comply to a standard bus structure, and the Ogre design synthesis system that enables library cores to be automatically interfaced and composed. Section II deals with the IP-XACT extensions and how they represent temporal behavior of IP cores with non-standard interfaces. Section III presents the synthesis techniques that leverage this information to automatically generate designs, and section IV summarizes the design productivity improvements observed in the development of software radios on FPGAs.

II. Meta-Data Descriptions: Adding Temporal Behavior Data to IP-XACT

IP-XACT provides most of the elements needed to describe cores. Its strengths and limitations are discussed in depth in [5][6]. In order to enable CAD tools to reason about the timing of cores and their interconnect in data-driven applications, extensions are needed to the basic IP-XACT schema. IP-XACT allows for external vendor extensions to support this type of extra information. This research extends IP-XACT in several ways, including extensions to describe the temporal behavior of library cores.

The design space in this research is constrained to designs modeled by the homogeneous synchronous data-flow model of computation [2]. Meta-data descriptions must be added to IP-XACT to describe the temporal behavior of cores that fitthis model. Three elements are added to IP-XACT via vendor extensions to do this: latency, data introduction interval and sample delay. Each of these extensions are represented by XML elements as extensions to the IP-XACT schema.

XML Code 1 This XML shows the vendor extensions added to IP-XACT to describe the temporal behavior of IP cores. This example defines a temporal SDF interface with a data introduction interval of 7, a latency 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>
```

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. XML Code 1 shows how latency is defined in XML. In the extension the latency is listed under the `<chrec:pipelineDepth>` element.

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. XML Code 1 shows the data introduction interval defined in XML listed as the value of the `<chrec:dataIntroductionInterval>` element.

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 ensure 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. Sample delay is especially important in systems that have feedback paths. A core with a sample delay of greater than 1 must exist in each loop in a design. In XML Code 1 the sample delay information is represented as the `<chrec:sampleDelay>` element.

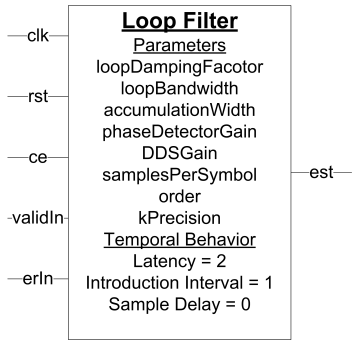

Fig. 2. This figure shows the loop filter block from the library. This block is modeled as having a latency of 2 and a data introduction interval of 1 and no sample delay.

Figure 2 provides an example of the top-level interface of the loop filter block. This example shows that the core takes

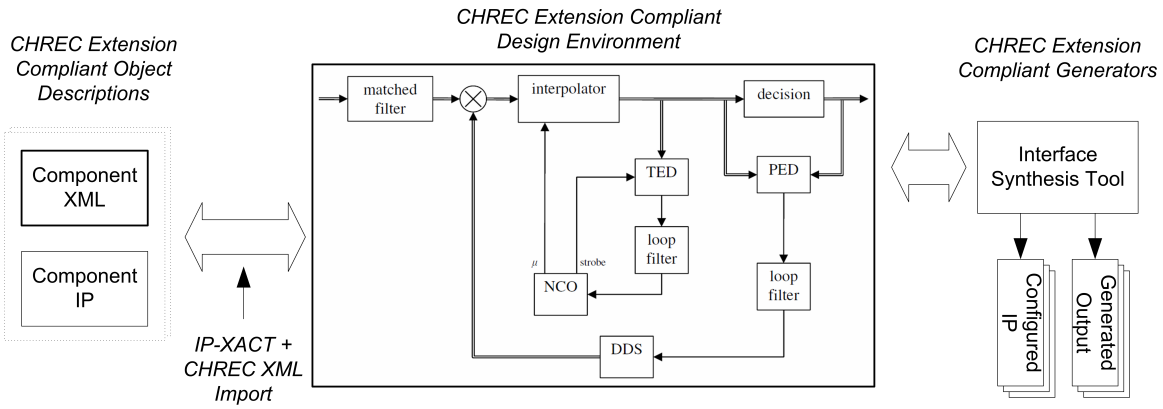

Fig. 3. The environment developed in this research imports HDL cores wrapped in XML, allows a designer to structurally connect these cores to one-another, uses this structure to synthesize a functionally complete design, and generates a valid HDL design which can then be synthesized using standard techniques.

two clock cycles to compute one result, it can accept new data every clock cycle, and there is no internal sample delay. This figure also shows the high-level of parameterization that is presented to the user.

III. Synthesis Techniques

There are two important aspects of the synthesis techniques developed in this research that help to improve reuse and design productivity. The first is a simple design environment that is easy for a designer to understand and use, and the second is a powerful synthesis tool-flow that allows designers to easily create valid designs from their specifications.

A. The Design Environment

The design environment that leverages the IP-XACT XML and the added extensions is summarized in Figure 3. The environment is connected to a library of parameterized IP cores which are wrapped in IP-XACT XML with the extensions described in Section II. This core description XML can be automatically generated through a GUI interface from VHDL cores with additional required meta-data being added by the user . Any core that conforms to the SDF model can have an XML description generated and be imported into the core library and used by the synthesis system. Many existing hardware development systems leverage libraries of cores; however, this library differs from previous models because the native library is itself extensible by definition. This extensibility simplifies core reuse by allowing *any* core to become a part of the native library and not simply requiring designers to add cores through a black-box interface.

After the library is populated, it is imported into the design environment and presented to the user as a hierarchical library based on the vendor, library, name and version of the core. The designer can drag and drop cores onto a design surface to instance them. This is done by leveraging the Simulink system and presenting the library to the user as a set of custom Simulink blocks.

Once blocks have been instanced on the design surface, they can be connected to each other and their parameters can be set.

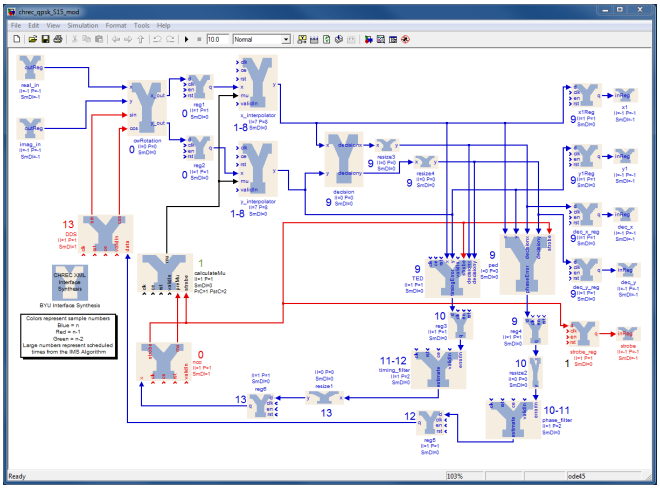

Fig. 4. This is the design window presented to the user. It allows the designer to instance cores and set their parameters as well as define the structural interconnect between cores.

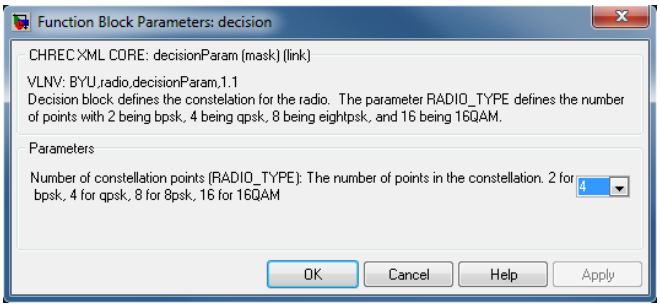

Fig. 5. This is an example of the parameterization window presented to the user. Parameters can be set in drop-down menus as shown here or can be typed in a text box.

Figure 4 shows a completed design on the design surface. The designer is only required to connect the data signals between cores and is not required to connect any control logic, clocking or reset circuitry. All control connectivity is automatically generated by the synthesis system.

Instances on the design surface can be parameterized to act in different ways. This parameterization is accessed by double clicking on the core instance, which displays a parameterization window as shown in Figure 5. The parameterization that this window shows reflects parameterization described in the XML core description.

The parameterization used in this research is also a contributing factor in increasing design productivity. In traditional hardware design, IP cores are parameterized at the low bit-width level. This research attempts to mask that level of parameterization by providing higher level parameters for the user to set which are then translated into the correct low-level parameter values for a core. The calculate mu block shown in Figure 2 shows examples of these higher level parameters. Parameters such as the loop damping factor and the phase detector gain are parameters that are specific to the software radio design domain and because these parameters are understood at a high level by radio designers, they increase productivity by providing a higher level of abstraction for the designer to reason with.

After the designer has set the parameters appropriately on each of the core instances, the design environment passes the complete structural design to the Ogre tool as shown in Figure 3. The Ogre tool completes the design by inserting control and glue logic into the circuit. The operation of this tool is described in the following section.

B. The Tool Flow

The Ogre tool uses the core description meta-data as well as the structural interconnection information from the design environment to construct a valid hardware design that can be implemented in an FPGA. The stages of this tools algorithm are described in general in Figure 6.

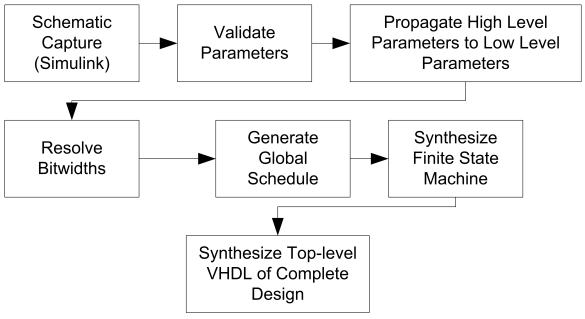

Fig. 6. This is the flow used by the Ogre synthesis system to generate complete designs.

Once the schematic capture of the design has been completed, the first step taken by the Ogre tool is to validate the parameters set by the designer. This is done by checking the parameters against their valid ranges or choice sets as defined in XML. Once all designer-set parameters have been validated, the high level parameter values are used in mathematical expressions to calculate the value of low-level parameters. An example of such a mathematical expression is shown in XML Code 2. These mathematical expressions are supported natively in the XPath language and can be quite expressive [7].

XML Code 2 This XML codes shows how high level parameters are translated into low-level parameters using mathematical expressions as defined by IP-XACT. This XML snippet is listed as a child of the low-level parameter whose value is being calculated.

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

Once both high and low-level parameters have all been resolved, bit-widths are propagated through the design. The bit-widths for the cores in the library are all dependent on parameters which relate the input bit-width to the output bit-width. The designer sets the bit-width on the input ports to the design and these bit-widths are then propagated through the design to set the bit-widths on each of the cores in the design and to calculate the width of the signal wires that connect them.

Following bit-width resolution, a global schedule for the design is generated. This schedule controls which cores are turned "on" or "off" at a particular time and it ensures that the order in which they are activated allows data to correctly flow through circuit to create the correct functionality. This step of the tool relies heavily on the temporal behavior information listed in the behavioral extensions to IP-XACT. The details of the algorithm used to create this schedule is beyond the scope of this paper. In general, however, this algorithm is done by creating a dependency graph from the circuit netlist and then creating a schedule using a variant of the iterative modulo scheduling algorithm described in [8]. The result of this schedule is a list of start and end times for each core in the design as well as an overall latency, in clock cycles, for the entire design.

Once a valid schedule has been created, it is implemented in a finite state machine (FSM). This FSM controls the operation of the circuit by manipulating the `dataValid` and `clockEnable` signals on the scheduled cores. The tools know the location of these signals because they are specified in the XML meta-data description. During the first clock cycle that a core is active, the `dataValid` and `clockEnable` signals are asserted. For every cycle following the first in which the core should be "on" the `clockEnable` signal is asserted. The completed FSM design is output to a VHDL file. The FSM implements the schedule and therefore ensure that all data dependencies are met in the design.

After the FSM has been generated, it is inserted into the

design and the control signals for the design are connected. Each of the `clockEnable` and `dataValid` signals are connected from the FSM to their respective cores. The global clock and reset signals are automatically connected to the cores and the correct core ports are wired to top-level input and output ports. This completed design is then also exported to a VHDL file. This VHDL file can then be used in traditional FPGA design flows to synthesize, place and route the complete design on to an FPGA fabric.

IV. Results: Building Software Radios

The meta-data descriptions and the synthesis techniques described in the previous sections were tested and demonstrated by building several software radios. The purpose of these tests was to demonstrate design productivity improvement by using the Ogre tools. To accomplish this, two versions of a QPSK receiver (see Figure 7) were designed by hand. The first was a combinational version of the receiver in which a single iteration of the loop ran in a single clock cycle. The design time for this radio was approximately one day. A pipelined version, multiple clock cycles for the loop, was also developed by hand and the design time for this radio was three days.

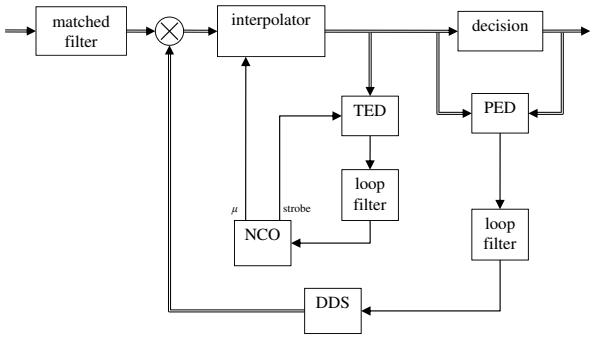

Fig. 7. A QPSK Receiver

To demonstrate design productivity improvement, each of these QPSK radios was also built using the Ogre system. The design time for the combinational radio, which was a day by hand, was reduced to less than an hour using the Ogre tools. The pipelined version was also implemented in Ogre and a functional radio was produced in less than an hour. In addition to the much shorter design time, the generated radio was more efficient; it took only 14 cycles to complete the loop where as the hand generated one required 15.

Design productivity improvements were also demonstrated by the ability of designers to develop several different radio personalities in a single afternoon. Using the entire Ogre reuse system, this research was able to produce seven different QPSK implementation which differed in their latency and area requirements, a BPSK design, and 8PSK and 16QAM designs in a single day. Each of these radios was implemented on an Xilinx XTremeDSP FPGA board and demonstrated to correctly produce a constellation.

The type of design time improvement demonstrated in this research 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.

V. Conclusion

This paper has presented a core reuse system that demonstrates the ability to increase design productivity in data-driven designs implemented in reconfigurable systems. This design productivity improvement has been obtained by leveraging meta-data descriptions of IP cores and using these descriptions in an end-to-end synthesis environment to allows a designer to rapidly implement designs and explore the design space.

IP cores are described in IP-XACT with extensions for describing temporal behavior. The parameterization provided by IP-XACT is also leveraged. These cores are imported into a library which is made available to the designer. This library is natively extensible and allows easy importation of cores that fit into the SDF model.

The synthesis system allows the designer to connect cores into a complete design and to set parameters on these cores to change their internal operation. The synthesis system also removes the need for the designer to specify the timing of the circuit by automatically generating a schedule and FSM to control the circuit. All other control logic is also automatically generated and inserted into the circuit.

The combination of parameterized IP cores, their meta-data descriptions, and the end to end design environment has enabled increases in design productivity. Designs that took days to build by hand were able to be built in a matter of hours. This design approach is particularly applicable to data-driven designs that are a natural fit for FPGA and other reconfigurable computing platforms.

References

[1] S. Hauck and A. DéHon, Eds., *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Morgan Kaufmann Publishers, 2008.

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

[3] R. Passerone and J. A. Rowson, "Automatic synthesis of interfaces between incompatible protocols," in *Proceedings of the 35th Design Automation Conference (DAC 1998)*, June 1998, pp. 8–13.

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

[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] N. Rollins, A. Arnesen, and M. Wirthlin, "An XML schema for representing reuable IP cores for reconfigurable computing," in *Proceedings of the National Aerospace and Electronics Conference (NAECON 2008)*, July 2008.

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

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