FROM REQUIREMENTS CAPTURE TO SILICON: A MODEL-DRIVEN SYSTEMS ENGINEERING APPROACH TO RAPID DESIGN, PROTOTYPING AND DEVELOPMENT USED IN THE OAK RIDGE NATIONAL LABORATORY'S COGNITIVE RADIO PROGRAM

Mark A. Buckner, (Oak Ridge National Laboratory, Oak Ridge, TN), Brian Kaldenbach (ORNL), Nory Nakhaee (Sundance DSP, Reno, NV), Don Bouldin (University of Tennessee, Knoxville, TN), Jonathan Mills (Indiana University, Bloomington, IN), Michael R. Moore (ORNL)

ABSTRACT

The performance and complexity of the signal processing hardware accessible to SDR/CR/RADAR designers has quickly out-paced the available design tools. The advances in Digital Signal Processors (DSP) both fixed- and floating-point, Field Programmable Gate Arrays (FPGA), and multi-core processors have enabled rapid prototyping and deployment of platforms that can be dynamically reconfigured in the field to implement a variety of SDR/CR/RADAR waveforms. Until recently the process of creating waveforms meant starting with high-level mathematical models and simulations and then creating production quality code that can operate on this variety of specialized hardware using either hand coding or vendor specific tools, which are typically limited to single processor solutions. This paper discusses an integrated model-driven design process and tool-flow used in ORNL's Cognitive Radio Program. It describes how the process and tool-flow are used on a variety of SDR and CR projects and in the development of a software-defined RADAR environment simulator. It describes how, from a single Simulink® model, a single deadlock free real-time multi-processor application is created and executed on a network of heterogeneous processors. We also describe recent progress on extending the process/tool-flow to design digital ASICs and our plans for future extensions. We close by highlighting the benefits being realized from applying this design flow to SDR/CR/RADAR projects at ORNL: 1) a significant reduction in the time required to develop, prototype, implement and test SDR/CR/RADAR waveforms, 2) increased reusability/retargetabilty of SDR/CR/RADAR designs and signal processing library components, 3) the ability to quickly port SDR/CR/RADAR waveforms to different hardware systems and processor types, 4) improvements in documentation, and 5) traceability of system components back to original requirements.

1. INTRODUCTION

The process of taking a radio or radar design from initial concept and requirements capture to final acceptance testing is fraught with many challenges and difficulties. In traditional development approaches once requirements are captured and models of the desired waveforms and signal processing algorithms are developed, the coding of the waveforms and algorithms for candidate hardware requires manual coding which is tedious, time consuming and error prone. In addition, as manual coding and refinement proceeds, the implementation often diverges from the original mathematical models, and it becomes increasingly difficult to assess the effect the differences will have on other components of the design at the system level. The situation becomes even more complex if multiple heterogeneous processors such as fixed- and floating-point DSPs, FPGAs and general purpose processors (GPPs) are used. Optimally partitioning the system across a mixture of heterogeneous processors requires special knowledge of the hardware and requires use of vendor specific tools. In this case additional design issues must be addressed including inter-processor communication, clock and data synchronization, loading/booting, memory constraints and processor specific languages (C, C++, assembly, VHDL, Verilog). Often, final trade-off design decisions must be based on performance data obtained from implementation in actual hardware, which requires rapid prototyping on candidate hardware. Combine these challenges with the added need for documentation, testing and requirements traceability and one begins to get a feel for how daunting SDR/CR/RADAR development really is.

Our team faced all the above challenges and more as we embarked on developing a new radar environment simulator (RES). Our RES is responsible for presenting to the radar under test a variety of operator-defined scenarios, which include radar returns from multiple simultaneous targets/objects and clutter sources (see Figure 1). The RES Manager runs on a workstation and provides system control functions and scenario generation. Scenario generation feature allows a RES operator to create detailed scenarios with all the required technical parameters available in a

library of environmental and target data. This library contains the best available models of projectile RCS (radar cross-section) and clutter (discrete and distributed, fixed and moving). Scenarios are viewable off-line, and the RES operator can pause, rewind, and fast forward the scenario during operation. Real-time signal/return generation is accomplished by a network of heterogeneous processors in the RES Processor Chassis. The RES will generate a simulated environment that exceeds the performance of the radar under test, which includes many simultaneous targets, and extends beyond the radar's maximum range.

The primary RES design goal is the creation of a *Simulink®-based executable simulation framework with real-time hardware acceleration on a network of heterogeneous COTS processors* with the following characteristics: 1) open systems architecture, 2) model-driven, 3) automatic multi-processor code generation for from a single Simulink® model, 4) use of modular reconfigurable COTS hardware, and 5) model/code re-use. While our initial evaluation of candidate hardware and design tools failed to identify a tool-set/design-flow and hardware that fully met the above criteria we found enough key parts of a solution that if combined would and so decided to lead an effort to combine them and fill in a few missing elements.

What follows is a description of where we are in the process.

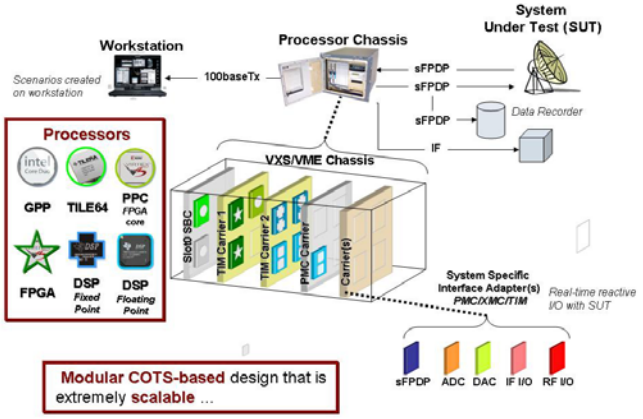

Figure 1 Diagram of RES components and Radar Under Test.

2. MODEL-DRIVEN APPROACH

The model-driven design approach provides an array of benefits including a system view of all code/components, a seamless testbench, requirements traceability, auto-code generation and a host of others.

The system-wide view – that is, block-diagrams or schematic capture views of all of the subsystems and their connections – gives developers a more comprehensive view of the system while providing well-defined interfaces for separating functions. This simultaneously reduces the complexity for individual tasks while fostering the coordination required for large teams to develop code jointly. This approach automates not only code generation but also interface definitions among subsystems.

The seamless testbench allows developers to start with subsystems that have well defined interfaces, while the functionality of those subsystems are still in their infancy. Thus, providing the infrastructure for end-to-end testing before the entire code has been generated. By providing on-screen scopes and data results, it also provides the developer with the immediate feedback required to reduce development time.

As discussed below, requirements traceability is another attribute that provides much greater accountability for top-down assessments of the status of the development.

3. TOOL-FLOW

In this section we highlight requirements capture and traceability using Rhapsody® and DOORS® and the code development process including both a discussion of how the tools are integrated as well as giving examples of the benefits of the model-driven approach. Figure 2 depicts our general design flow which was adapted from the Telelogic Harmony™ Process.

3.1. Capturing & Tracing Requirements

Use Cases Analysis is used to gather, analyze and capture requirements that are tracked using the Rhapsody® Gateway and DOORS®. In addition to common Use Cases, additional UML diagrams are used to document requirements (Object Diagrams, Sequence Diagrams, Collaboration Diagrams, State Charts, and Flow Diagrams). Requirements traceability and coverage are tracked using a combination of DOORS® and Rhapsody® Gateway.

This provides both the developers as well as the users the ability to track and understand the progress and status of the project. It also helps bridge the communication gap that typically hampers discussions between developers – who by necessity are deep into the details of the signal processing – and users/customers that have more of a black-box view of the system.

3.2 Modeling & Simulation

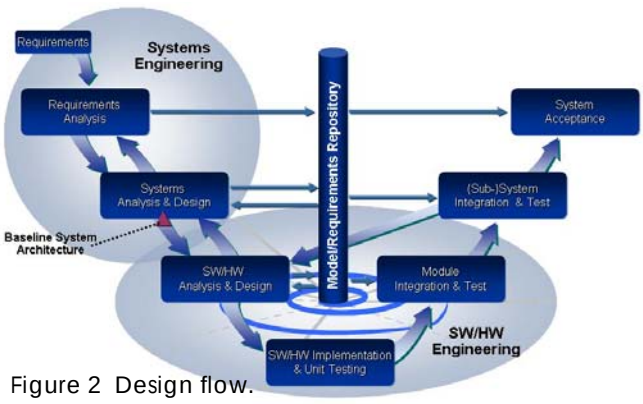

Figure 2 Design flow.

All models are created using Simulink®. This provides the seamless testbench and developer environment that crosses all DSP target devices, as well as providing scopes and other data viewing windows that give developers the immediate feedback they need.

3.3 Partitioning and Assignment of Tasks using PARS

PARS is a companion toolbox for The MathWorks™ Simulink® enabling the generation of a multi-DSP/FPGA application from a single Simulink® model. PARS was originally designed by Sundance as part of a development contract for the US Navy. At ORNL's request, Sundance is adding additional capabilities to support the RES development. These capabilities include 1) support for Simulink® HDL Coder™, 2) support for Real-Time Workshop® Embedded Coder™ (RTW-EC), which enhances real-time signal processing capabilities and efficient compact code generation, 3) implementation of profilers that allow access to low-level clock timing for DSP and FPGA tasks and 4) MEM blocks that allow developers to instantiate look-up tables necessary for the RES.

After the application model is designed and verified in the Simulink® environment, the developer uses PARS to aggregate and assign subsystems to the desired processor elements in the target hardware system, DSP (fixed- or floating-point) or FPGA of the host processor. Multiple tasks/subsystems may be assigned to a single DSP or FPGA. MATLAB/Simulink® users can quickly design and simulate platform-independent models and then automatically generate code to rapidly prototype on candidate multi-DSP, multi-FPGA hardware.

Another critical element in the RES design is the ability to create and access multiple large look-up-tables. PARS MEM blocks allow the user to manipulate large amounts of data in static arrays at run-time. PARS MEM blocks can be implemented on both DSP and FPGA tasks. During user task operation, the protocol parser is able to access the contents of memory arrays. With added calls within the user task loop, it is also able to synchronize access to the static memory arrays. The generated code for PARS MEM block is automatically synchronized with code generated by Simulink® for the developer's model. By utilizing PARS MEM block, developers can download/upload a section of memory inside a DSP or FPGA at any time irrespective of the state of the underlying task. For example, the memory can be uploaded when the task is waiting for input, processing or waiting for output.

The following operations can be performed transparently for I/O and data processing: 1) initialize data block (at compile/link time), 2) write data block (at run-time), 3) read data block (at run-time) and 4) synchronize processing with static array access (at run -time).

3.4 Code Generation

Auto-code generation is the biggest potential time-saver for developers but also has had to overcome a decade or more of being oversold to the community. The tools described in this paper go beyond providing programmers with subroutine skeletons and naming conventions to actually provide the complete code required to produce the desired functionality. With recent advances in the quality of the tools, they have also addressed historic concerns regarding their ability to handle real-time signal processing and limited computational resources.

Automatic code-generation requires that a restricted subset of available blocks/tools (libraries) be utilized. While this was a more severe restriction in recent years, the rapid increase in available libraries significantly reduces the impact. Developers have to be aware of the extent of available libraries and "toolboxes" and make sure that they acquire all appropriate libraries. This exercise alone significantly improves development time by providing developers with the greatest functionality in the shortest time.

PARS auto-generates the final code by calling target-specific tools, such as RTW-EC, Xilinx® System Generator, HDL Coder™, and Code Composer Studio™ (CCS). RTW-EC generates C and C++ code optimized for embedded systems from Simulink®, Stateflow®, and embedded MATLAB® models. Simulink® HDL Coder™ generates bit-true, cycle-accurate HDL (VHDL and Verilog) from Simulink®, Stateflow®, and embedded MATLAB®. CCS generates C and C++ code optimized for Texas Instruments DSPs.

3.5 Code Compilation and Loading

PARS compiles a single application by invoking the Diamond tools from 3L which include Diamond DSP and Diamond FPGA. Diamond is an optimized microkernel RTOS based on Communicating Sequential Processes (CSP) and has been the de facto parallel processing RTOS from the time of Transputers [4]. Diamond FPGA has the ability to encapsulate FPGA cores as tasks within its process flow. This means that interfaces between FPGA and DSP tasks are now completely ubiquitous and transparent [1].

Diamond configurer uses the PARS generated configuration file and tasks to map the tasks onto processing elements and task communications channels on physical connections.

3.6 On-Hardware Task Performance Profiling

Our first objective is to create a single complete Simulink® model running on a host processor to verify functionality and requirements compliance, with the understanding that this instantiation of the model will not meet the real-time performance requirements. Once we are assured that the Simulink® model is functionally accurate, we begin a process of profiling and timing analysis to determine the processing time and memory consumed by the various elements of the model. We use this information to iteratively repartition the model to meet final performance and timing requirements.

PARS is capable of profiling multiple parallel tasks during runtime, whether those tasks are assigned to a DSP or FPGA. On demand PARS generates suitable code to extract low-level timing information during run time. This timing information is then sent to the host computer via a synchronized channel so that transmission does not affect the timing of the tasks.

Table 1 shows an example of profiling statistics obtained for a sample model. They are accurate according to the actual time taken on hardware. PARS-Profile is available for both DSP and FPGA tasks.

Table 1 Example performance data obtained PARS profiling (Type O: output, I: input, P: process).

Task	Type	Processor	Clock Frequency (cycles/sec)	Time (sec)	Calls	Time/Call (sec)
DSPinput	I	ROOT	1000000000	1.40826568	11	0.128024153
DSPinput	P	ROOT	1000000000	0.000013888	11	0.000001263
DSPinput	O	ROOT	1000000000	0.000006632	11	0.000000603
DSPoutput	I	ROOT	1000000000	1.407937192	11	0.12799429
DSPoutput	P	ROOT	1000000000	0.000004216	11	0.000000383
DSPoutput	O	ROOT	1000000000	0.000011728	11	0.000001066
CalcSignalDelay	I	FPGA1	50000000	1.94540186	12	0.162116822
CalcSignalDelay	P	FPGA1	50000000	0.00000024	12	0.00000002
CalcSignalDelay	O	FPGA1	50000000	0.00000044	12	0.000000037

3.7 Configuration Management and Version Control

Subversion, an open source version control system, is used to provide version control and tracking for all elements of the RES incuding documentation, models, and code.

3.8 HWIL Code Verification

Hardware-in-the-loop (HWIL) methodologies are an inherent part of the above-described design process. That is, by targeting actual computing elements with individual components of the Simulink® model, the model actually runs on the target system (as opposed to a general purpose processor.) There are several advantages to this approach, 1) the model is bit-true and cycle-accurate to the final design, 2) target resource limitations (both clock rate and gate count) are dealt with during the model design process and 3) the models run at real-time signal processing speeds, which is typically 10^4 or more faster than general purpose processors running the same models.

3.9 Documentation

Automatic documentation of designs and test results are generated using a combination of Simulink® Report Generator™, MATLAB® Report Generator™ and Rhapsody® and DOORS®.

4. RESULTS

ORNL has successfully worked with key vendors to integrate their tools into a complete end-to-end model-driven design and simulation framework for SDR/CR/RADAR waveform design. The current framework includes re-use of legacy code, auto-generation, compilation, and creation of a single real-time executable application, task profiling, and baseline performance demonstrations on multiple DSPs and FPGAs in prototype hardware. In the case of RES, baseline performance data is being gathered from actual hardware and will be used to determine the number, type and configuration of processors for the final system. Work is ongoing to further improve and extend the capabilities of the process and design flow. A few of these are highlighted in the next section.

5. FUTURE EXTENSIONS

There are several ongoing tasks to further improve and extend the tool flow to enable support for the following:

- PICO E-15 FPGA CardBus card [6]
- Co-simulation and code generation for Power PC cores and digital fabric/cores within Xilinx FPGAs
- Tilera TILE64™ Processor [7]
- Digital ASIC design
- Field Programmable Analog Arrays (FPAA)
- Field Programmable Neuron Arrays (FPNA)
- Extended Analog Computer (EAC)

5.1 PRELIMINARY ASIC RESULTS

UTK, in collaboration with ORNL, has successfully synthesized a digital filter example using VHDL generated by HDL Coder for both a Xilinx FPGA using Synplicity Synplify Pro® and an ASIC process using Synopsys.

Pre-synthesis simulation was performed using ModelSim and post-synthesis with ModelSim was pending at the time of writing.

The design uses about 5% of a Xilinx xc2v500fg256-6 which is roughly 25K system gates (See Figure 3 FGPA layout).

The ASIC implementation in the TSMC (Taiwan Semiconductor Manufacturing Company) 180-nm process takes 4146 standard-cells total; 47K pairs of fets or 94,218 fets/4 = 23K nand-gates. The ASIC layout consists of 4193 components arranged into 80 rows and uses four of the available 8 layers of metal for routing (see Figure 4 ASIC layout). The area consumed is 1,126,013 x 851,140 db units or approximately 11.26 mm x 8.5 mm = 95.8 mm^2.

Eventually, the design will be combined with an existing VHDL core and taken through to an ASIC layout and post-layout simulation, but the results achieved so far establish the flow to ASIC as feasible.

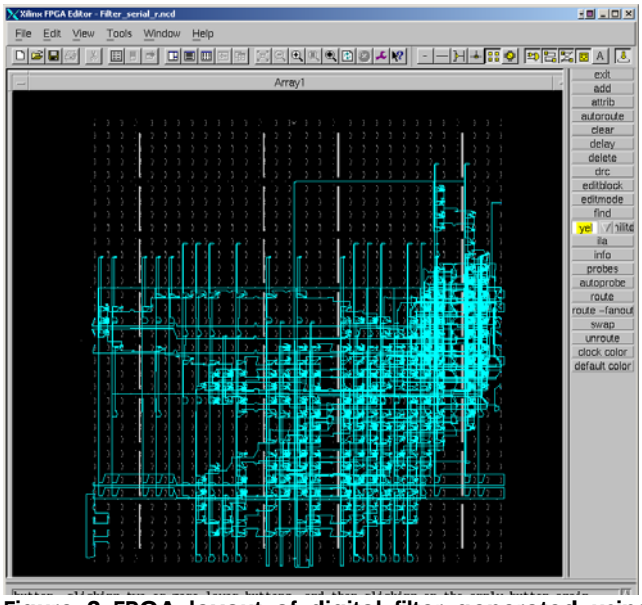

Figure 3 FPGA layout of digital filter generated using HDL Coder.

5.2 FPAA/FPNA/EAC

Work is underway to extend the current tools and design flow to be able to target FPAAs [9], FPNAs [10] and EACs [11],[12].

FPNAs based on floating-gate technology enable one to rapidly design and prototype analog circuits in much the same way FPGAs have enabled digital circuit design and prototyping.

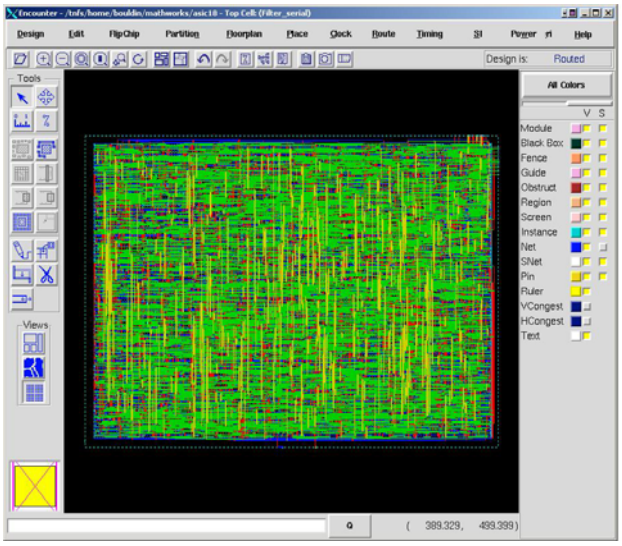

Figure 4 ASIC layout of digital filter generated using HDL Coder for TSMC 180 nm process.

While similar to FPAAs, FPNAs utilize reconfigurable analog blocks designed to mimic biological neurons (soma, axons and dendrites) and enable the implementation of biomimetic computation and signal processing, which is implicit and inherently parallel in nature.

The EAC accelerates computation using implicit functions inherent in the materials from which it is fabricated. According to Rubel, the EAC is a family of special-purpose devices that computes specific functions by analogy: heat conducting plates, soap films, vibrating strings and membranes, plasmas, slime mold, neural tissue, DNA and so forth [13]. All of these connect a user's problem to a natural object that solves it by means of a specific analogy.

Generally-applicable EACs [13] have been built that are useful in a wide range of application domains. Their generality requires users to draw an analogy between the EAC and its applications that divides its operation into two parts: (1) the EAC configuration, which constrains the properties of matter and energy that perform the computation, and (2) the meaning ascribed to those physical processes. Each of these two parts is necessary to define computation by analogy [11]. By ascribing problem-specific meanings to an EAC configuration, that configuration may solve many different problems. For example, one EAC configuration, unchanged, can compute butterfly wing morphogenesis, aircraft recognition, and small instances of the NP-complete problem Hamiltonian cycle.

Making the capabilities these technologies have to offer readily accessible to researchers and engineers will radically change the performance and capabilities of SDR/CR and RADAR systems of the future.

6. CONCLUSIONS

The benefits realized to date by development and implementation of the integrated model-driven design process and tool-flow described above include: 1) a significant reduction in the time required to develop, prototype, implement and test SDR/CR/RADAR waveforms, 2) increased reusability and retargetability of SDR/CR/RADAR designs and signal processing library components, 3) the ability to quickly port SDR/CR/RADAR waveforms to different hardware systems and processor types, 4) improvements in documentation, and 5) traceability of system components back to original requirements.

While the benefits are significant, one thing to keep in mind is the process described in this paper is a departure from more traditional design approaches and as with any new approach, there is a learning curve associated with adopting and implementing it. But, it is our opinion that the benefits far outweigh the cost of learning and implementing this approach.

The importance of the relationship between the user requirements, the RES developers (led by ORNL) and the tool provider (Sundance) cannot be overstated. The dynamic interaction between these three entities resulted in a complete end-to-end tool that is already resulting in a paradigm shift in the way that complex real-time signal processing challenges are tackled.

6. REFERENCES

[1] D. Neumann, Sr Principal Engineer and C. Nelson, DSP Engineer, U.S. Navy, SPAWAR Systems Center - Charleston, N. Nakhaee, Chief Executive Officer and B. Vacaliuc, Chief Technology Officer, *Sundance DSP Reconfigurable Approach Wins for Adaptive Beamforming* http://www.cotsjournalonline.com/home/article.php?id=100647&pg=1

[2] M. Ahmadian, N. Nakhaee, and A. Nesterov. *Rapid Application Development (RAD) and code optimization technique.* Global Signal Processing Conference (GSPx), 2004.

[3] M. Baleani, A. Ferrari, L. Mangeruca, A.L. Sangiovanni-Vincentelli, U. Freund, E. Schlenker, and H.J.Wolff. *Correct-by-construction transformations across design environments for model-based embedded software development.* Proceedings of Design, Automation and Test in Europe, 2:1044 – 1049, 2005.

[4] Information on Diamond RTOS for Sundance Products is available from the 3L Web site: http://www.3l.com/Diamond/Sundance/diamond_for_sundance.htm

[5] Subversion Project Wed site: http://subversion.tigris.org/

[6] Pico Computing, Inc. Card Product Web site: http://www.picocomputing.com/products/cards.php

[7] Information on about the TILE64™ PROCESSOR FAMILY is available on their Web site: http://www.tilera.com/products/processors.php

[8] C.M. Twigg, P.E. Hasler, I.F. Baskaya: A Self-Contained Large-Scale FPAA Development Platform. ISCAS 2007: 1187-1191

[9] C. Gordon, E. Farquhar, and P. Hasler, "A family of floating gate adapting synapses based upon transistor channel models," in *Proceedings of the 2004 International Symposium on Circuits and Systems*, 2004. ISCAS '04., vol. 1, May 2004, pp. I–317–I–320.

[10] E. Farquhar, C. Gordon, and P. Hasler, "A field programmable neural array," in *Proceedings of the International Symposium on Circuits and Systems*, Kos, Greece, May 2006.

[11] J. Mills. The architecture of an extended analog computer core. *Fourth Workshop on Unique Chips and Systems (UCAS-4)*, April 2008.

[12] J. Mills, (in press). The nature of the extended analog computer. *Physica D* (to appear July 2008).

[13] A. Adamatzky et al., *Unconventional Computing 2007*. Luniver Press (2007).

[14] B. Himebaugh, Design of the EAC, www.cs.indiana.edu/~bhimebau (2005).