System-Level Tools to Accelerate FPGA Design for Signal Processing

Jim Hwang
Sr. Manager, DSP Software and Design Methodologies
Why FPGA DSP?

• High performance
• Flexibility
• Time to Market
• Functional extensions to existing equipment
• Standard part (no NRE/Inventory issues)
• Early system bring-up on hardware
The Highly Parallel Signal Processor

- Up to 12.5 million gates
- Embedded RISC CPU
- Programmable I/Os with LVDS
- 3.125Gb Serial
- Programmable Fabric (300+ MHz)
- Synchronous Dual-Port RAM
- BRAM
- CLB, IOB, DCM
- 300MHz pipelined 18b x 18b multiplier
Exploiting Parallelism: Conventional DSP Solutions

- New DSP architectures such as VLIW and super-scalar have one goal: provide higher degrees of parallelism.
- Architecture evolution along this design axis does not scale.
  - Too many MAC functional units makes programming, compilers and scheduling difficult.
- The effective computing per chip area decreases.
  - Memories grow geometrically while the datapath does not.
DSP Systems in FPGAs

• Device technology is only part of the solution
• The software and IP are complex, and historically have been a barrier to entry
• Require design methodologies for
  – Productivity
  – Rapid design exploration
  – Hardware abstraction
• Single source for the entire design & development cycle
  – Modeling
  – Verification
  – Implementation
  – Automatic code generation
FPGA as DSP Platform

• Like other ASICs, the FPGA is largely a value proposition for very high performance applications
  – Digital communications infrastructure
  – Software defined radio
  – Video & imaging
  – SAR, adaptive arrays & beamforming
  – Spartan-II family devices counter this trend

• This has profound implications for design methodology
  – DSP & microprocessors are very good at tasks where performance is not a problem
  – Not targeting (most of) the tera-bytes of legacy DSP code (yet)
What Language?

• Do you use MATLAB?

• Why do we like it?
  • Easy to learn
  • Interpreted
  • High level abstractions
  • Extensive libraries and built-in functions
  • Rich facilities for data analysis and visualization
  • Used in many signal processing textbooks

• But is it a good language for specifying hardware?
  • Imperative language with sequential semantics
  • No concurrency model
  • Dynamically typed (flexible, but…)

{System}C{++}? 

- Considerable activity advocating imperative, sequential languages for system level design
  - C, C++, SystemC
  - Co-Ware, Synopsys, Cadence
- This is not a bad thing for
  - Embedded systems
  - High-level (e.g. untimed, untyped) functional modeling
  - Validation of complex systems
- This is not a good thing for
  - Hardware description
  - High-performance DSP system design
{System}C{++}?++

• Language designers give a lot of thought to semantics
  – C/C++ semantics derived from microprocessor considerations
  – A good language for hardware must model concurrency
  – Object oriented principles are not a cure for semantic flaws

• Hardware synthesis (from C) is not a solved problem
  – High-performance circuits carefully tuned to target technology
  – All “C”-based design flows depend on design iteration to the point of “RTL” code before synthesis
  – At this level of abstraction, C/C++ becomes contorted
  – VHDL may not be beautiful, but it models
A Fallacy¹

• Premise: “software” is easier than “hardware”, consequently, systems should be specified in the language of software engineers

• Empirical evidence to contrary
  • Software products invariably ship with more bugs than hardware products
  • There are more software engineers at Xilinx than hardware engineers

• Conclusion: do not assume that (imperative, sequential) “software” languages are best suited for DSP hardware and system specification

¹Observed by Bob Broderson (UCB)
A Picture is Worth…

- Classical DSP algorithm description
  - Block diagrams
  - Signal flow graphs
  - Inherently concurrent

- Visual languages and development environments
  - Synchronous Data Flow
  - Ptolemy (UCB)
  - SPW (Cadence)
  - Simulink (MathWorks)

- A good match!
Simulink

• Graphical simulation environment
  – Continuous and discrete time dynamical systems
  – Well suited for modeling hardware (and getting better)

• Block libraries for DSP, communications, image processing, digital control, and much more

• Open architecture
  – Extensible
  – Public APIs
  – Amenable to programming in C, C++, Java, …

• MATLAB inside (and underneath)
  – The implications should not be underestimated
System Generator for DSP

- Xilinx software for FPGA modeling & implementation
- FPGA interfaces provided in Simulink environment
  - Libraries of functions for modeling DSP (and other) systems
  - Automatic code generation of FPGA circuits
  - Fast on-ramp into the FPGA
- System level abstractions create new opportunities in the lab
  - Explore architectures for DSP algorithms
  - Implementation issues (e.g. quantization, pipelining)
  - Emphasize system level test and test bench methodologies
  - Actually run the system in silicon!
Advanced Hardware Lab
Circa 1984

Got wirewrap?
Advanced Hardware Lab
2002

Got System Generator?

xc2v2000e FPGA
System Generator for DSP

- Visual data flow paradigm
- Polymorphic block libraries
- Bit and cycle true modeling
- Seamlessly integrated with Simulink and MATLAB
  - Test bench and data analysis
- Automatic code generation
  - Synthesizable VHDL
  - IP cores
  - HDL test bench
  - Project and constraint files
System Generator (cont.)

- Supports common Simulink idioms
  - Data type propagation
  - Polymorphic blocks
  - Sample time propagation
  - Block customization
  - MATLAB hooks
SysGen Modeling: Analysis

- Observing quantization effects
  - All fixed point data carries floating point values as well
SysGen Modeling

- Automatic test bench generation for HDL simulation
  - Behavioral
  - Post-mapping
  - Post place & routing
- Log test vectors at input and outputs
- Pull into generated HDL test bench
A Simple MAC Engine

- Use MATLAB to customize the data path

- Select precision to avoid overflow
- Minimize resources required
- Parametric implementation
- Fine-tune the accumulator to avoid stall

\[ AccumWidth = m + \left\lfloor \log_2 \sum |h_i| \right\rfloor \]
CORDIC Processor

- Versatile family of algorithms for computing functions
  - Arctan, square root, division, logarithm, ...
- Shift-and-add architecture
- Well-suited to FPGA implementation

Iteration Equations

\[
\begin{align*}
x_{i+1} &= x_i + y_i \delta_i \\
y_{i+1} &= y_i - x_i \delta_i \\
z_{i+1} &= z_i + \delta_i \tan^{-1}(2^{-i}) \\
y &\to 0 \\
\delta_i &\in \{-1, 1\}
\end{align*}
\]
CORDIC Processor (cont.)

- Output precision depends on the number of PEs
  - Trade area for precision
- Use Matlab to parameterize the processor
  - Propagate parameters down to blocks
    - Data width
    - Constant values
  - Conditionally and iteratively instance PEs
LMS Adaptive Filter

\[ Y_{n+1} = X_n^T W_n \]

\[ W_{n+1} = W_n + 2\mu\varepsilon_n X_n \]

Sum-of-product calculations can be computed in parallel
Pipelined LMS Algorithm

- De-couple the LMS update and FIR PEs to exploit maximal pipelining

Pipeline balancing delays between LMS & FIR PEs
LMS Adaptive Update

- Single complex multiplier services filter taps
  - Maintains its own data stream to de-couple from filter data path

\[ W_{k+1} = W_k + 2\mu\varepsilon_k X_k \]
Example will consider implementation of the equalizer and carrier loop.
System Generator Model

Transmitter Model
- 16-QAM Source
- Matched Filter
- Sample rate Change

Passband Adaptive Equalizer
- Fractionally spaced (T/2)
- Polyphase decimator structure
- LMS coefficient update at the symbol rate

Carrier Recovery
- CORDIC based PE

Instrumentation
Embedded DSP

- The FPGA value proposition for DSP is predominantly high-performance applications
  - This does not mean retargeting legacy C code for low performance applications
  - Some value to be had with end-of-life low end DSPs
  - Virtex-II Pro PPC and MicroBlaze will play a big role, largely as controllers, at least in the near term

- System design will not begin with C programs
  - Kurt Keutzer in EE Times, 12 Sept. 2002:

A successful programming model may or may not be a conventional programming language, and the target device may or may not be an assemblage of conventional processors. The use of Matlab and Simulink in the Xilinx System Generator for DSP presents an interesting alternative for programming FPGAs. The key point is that the entire underlying device may be programmed in a single integral fashion. Software development environments using attractive programming models are certain to become differentiating features among programmable platforms. The "software-development environment as afterthought" era of integrated circuit design is rapidly drawing to a close.
Embedded DSP (cont.)

• The key problem to solve is not inferring the HW/SW cut-set
  – It is defining the right idioms and mechanisms for the heterogeneous specification of a complex system
  – This includes programming languages like C, C++, Matlab, SystemC, and Java, as well as VHDL and Verilog

• Simulink is the system level language and control panel
  – Modeling: design capture, simulation, refinement, functional validation
  – Hooks to programming languages for specification and implementation
    • Matlab, C++, C, Java

• System Generator is the means for creating the FPGA implementation
  – Fabric, multipliers, memories, embedded processors
  – Increased level of integration with embedded tools in the future
Using SysGen to Create CoreConnect™ Peripherals
SysGen Peripheral Overview

• Subsystem slave interface to the OPB
  – Separate in/out interfaces better depict data flow direction in Simulink
  – Abstract bus connections using global from/goto blocks
  – Acknowledgement generation logic

• Memory mapped register/FIFO interface to the System Generator DSP data path

• Address decoding (the basics)
  – Parametric peripheral select subsystem
  – Comparator-based re/we subsystem
Modeling Bus Transactions
The Future

- FPGA as platform for integrated DSP systems
- Design methodologies to support the platform
  - Abstractions that permit working in the language of the problem
  - Enable effective integration of re-usable components (cores)
  - Heterogeneous simulation and deployment options, accessible from a single system level environment