From Streaming Models to Hardware and Software Implementations

Kaushik Ravindran and Hugo Andrade
National Instruments Corporation, Berkeley, CA, USA

Workshop on Software Synthesis
October 14, 2011
National Instruments: What We Do

- Low-Cost Modular Measurement and Control Hardware
- Productive Software Development Tools
- Highly Integrated Systems Platforms

Used By Engineers and Scientists for Test, Design and Control
LabVIEW: Graphical System Design

High-Level Design Models

Dataflow | Configuration | Textual Math | Simulation | Statechart
---|---|---|---|---

- **Dataflow**
- **Configuration**
- **Textual Math**
  ```
  1 c = 0.285 + 0.013i;
  2 [X Y] = meshgrid(x, y);
  3 z = X + iY;
  4 for k=1:30
  5   z = z.*2 + c;
  6 end
  ```
- **Simulation**
- **Statechart**

**LabVIEW**
Graphical Programming

Desktop Platform
- Linux
- Macintosh
- Windows

Embedded Platform
- Real-Time
- FPGA
- MPU
LabVIEW Targets

- Scalable from distributed network to sensors
Outline

• DSP Designer framework

• Models, analysis, and exploration

• Deployment challenges

• Summary
Motivation

Application trends
• 1000’s of parallel tasks
• Large node/channel counts
• High performance requirements
• E.g. streaming DSP applications

Platform trends
• 100’s of processing elements
• Heterogeneous processors and memories
• Distributed I/O
• E.g. Heterogeneous FPGA targets
Streaming Model of the OFDM Transmitter

- **Nt = \{1, 2, 4\}**
  - Compile time - # transmitters
- **Nu = \{72, 180, 300, 600, 900, 1200\}**
  - Initialization time - Bandwidth
- **CP mode = \{'Normal', 'Extended'\}**
  - Run time
  - To overcome Inter-symbol-interference
  - Can be applied at symbol boundary
- **CP Vector**
  - Vector elements must be applied at symbol boundary
  - Vector selection based on CP mode

**Challenge:** How to express a domain expert’s algorithm specification in a model that is viable for analysis and implementation?
DSP Designer

• Development environment for creating streaming high-performance RF and DSP applications for FPGA targets

• Driving applications
  – Spectral analysis, signal intelligence, software radios

• Platforms
  – Heterogeneous FPGA platforms (R-series, FlexRIO)

• Target users
  – RF and DSP domain experts
Modeling System-Level Designs

New modeling constructs
• Systems
• Targets
• Mixed model diagrams
• Asynchronous Wires
LabVIEW DSP Designer
LabVIEW DSP Designer

LabVIEW VIs

Xilinx CoreGen Blocks

Data Ports
LabVIEW DSP Designer

- Auto buffer sizing to minimize resources
- Calculated firing counts and timing data
- Throughput constraints
LabVIEW DSP Designer

Calculated Schedule View
Value of DSP Designer
RF Design Flow

LabVIEW FPGA

FlexRIO

RIO Target

Peer-to-Peer Streaming

Transfer
Outline

• DSP Designer framework

• Models, analysis, and exploration

• Deployment challenges

• Directions ahead
DSP Designer Focus Areas

• DSP Diagram model of computation

• Analysis and optimization back end

• Performance models and timing library

• Actor definition

• IP modeling and integration

• Simulation and verification

• Code generation and implementation
## MoCs for Streaming Applications

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Expressive</th>
<th>Analyzable</th>
<th>Deterministic?</th>
<th>Bounded data rates?</th>
<th>Deadlock and boundedness decidable?</th>
<th>Static scheduling?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kahn Process Networks</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Integer Dataflow</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Boolean Dataflow</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SHIM</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Parameterized Dataflow</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Cyclo-static Dataflow</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Homogeneous Dataflow</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

*Key trade-off:* Analyzability vs. Expressibility

Analysis and Optimization Features

• Core dataflow optimizations
  – Model validation (deadlock and unboundedness detection)
  – Throughput and latency computation
  – Buffer size optimization (under throughput constraints)
  – Schedule computation

• Hardware specific optimizations
  – Resource constrained schedule computation
  – Retiming and fusion
  – Rate matching
  – IP interface synthesis

Analysis determines at compile time that this application executes in bounded memory and is deadlock free.
Buffer Size Optimization

Example SDF Graph

Repetitions vector: (3, 2, 1)

Problem Assumptions

• Execution times: (A,1), (B,2), (C,2)
• No resource constraints

Pareto Space of Throughput and Buffer Sizes

Exploration results in a Pareto space of throughput and buffer sizes

Resource Constrained Scheduling

Application Model

Micro-Architecture Model

Direct implementation:
Resources: 6 (*), 2 (+), 2 (-), 1 (<)
Latency: 4 cycles
Throughput: 1 sample / 1 cycle

Constrained implementation:
Resources: 1 (*), 1 (+), 1 (-), 1 (<)
Latency: 7 cycles
Throughput: 1 sample / 7 cycles

FPGA Based Soft Multiprocessor Systems

• Designer customizes multiprocessor architecture
  – Number of processors
  – Interconnection network
  – Custom co-processors

• Advantages
  – FPGA becomes accessible to software developers
  – Software compilation faster than hardware synthesis
Designing Soft Multiprocessor Systems on FPGAs

- Modern FPGAs provide the capacity to build a variety of micro-architectures
  - 50 processors – growing with Moore’s law
  - Complex memory hierarchy
  - Heterogeneous interconnection schemes
  - Custom co-processors for critical operations

Given a target application(s):
What is the best multiprocessor micro-architecture on the FPGA?
Optimization Toolbox

Application model (with cost annotations)

Heuristic list scheduling
Graph partitioning
Integer linear programming
Constraint programming
Simulated annealing

Toolbox of optimization methods

Implementation and Performance Analysis
Outline

• DSP Designer framework

• Models, analysis, and exploration

• Deployment challenges

• Directions ahead
IP Integration

Streaming Model of the OFDMA Transmitter

User HDL

Hardware IP

I/O with throughput requirements

Design Challenge: Generate a viable implementation of the OFDMA transmitter on a Xilinx Virtex-5 FPGA that meets correctness and performance requirements
Timing Models for IP Blocks

Rational Resampler

Execution time: 20
Initiation interval: 8

- Execution time: time to complete one firing, i.e. read 2 samples and produce 3 samples
- Initiation interval: minimum time between start of successive firings

Example: Source-Resampler System

S outputs 1 sample every 2 cycles

R reads 3 inputs samples in consecutive cycles and outputs 2 samples in the 2\textsuperscript{nd} and 3\textsuperscript{rd} cycles

Objective: Create a valid composition of these blocks so that behavior and timing constraints are respected
Source-Resampler: Interface Signals

Connecting $v_s$ output of Source to $u_r$ input of Resampler results in an invalid composition
Source-Resampler: System Implementation

Glue design problem: Given a set of actors, synthesize a glue (buffers and controllers), such that the resulting system satisfies correctness and performance properties.

Optimal throughput: 2 samples / 6 cycles
Optimal buffer size: 2
Solutions to the Glue Design Problem

• Solutions at the HDL and FSM level
  – Suffer state space explosion problem
  – Infeasible for most practical systems

• Solutions based on abstract models
  – Enable efficient static analysis
  – Support automated synthesis

• But important to choose the right abstractions that yield correct and non-defensive implementations
Source-Resampler: Static Dataflow Model

Self-timed schedule to achieve optimal throughput
(requires buffer of size 5)

Limitations of the SDF model:
• Does not capture how tokens are accessed
• Analysis conservatively allocates space for tokens from firings of S that occur while R executes
• Resulting implementation is defensive

Cyclo-Static Dataflow Model

Self-timed schedule to achieve optimal throughput
(requires buffer of size 1)

Limitations of the CSDF model:
• Does not capture requirement that Resampler IP must receive 3 tokens in consecutive cycles
• Analysis underestimates space needed for the buffer
• Resulting implementation is incorrect
Static Dataflow Model with Access Patterns

**Self-timed schedule to achieve optimal throughput**
(requires buffer of size 2)

Strengths of the SDF-AP model:
- Explicitly specifies how tokens are consumed and produced in time
- Analysis yields a buffer of size 2
- Resulting implementation is correct and non-defensive
Access Pattern Example

\[ [1,0,1,0,1,0,0,0] \quad \text{ex. 8} \quad [0,0,0,1,0,0,0,1] \]

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 2 & 3 & 1 & 4 & 5 & 6 & 7 & 8 \\
\end{array}
\]

consume
execute
produce
Case Study: OFDMA Transmitter

- Glue design using SDF-AP models
  - Improves buffer sizes compared to SDF
  - Results in a non-defensive controller implementation
Outline

• DSP Designer framework
• Models, analysis, and exploration
• Deployment challenges
• Directions ahead
Directions Ahead

• Model and analysis extensions
  – Efficient methods to check correctness and non-defensiveness
  – Automatic characterization of access patterns
  – Formalize relation between abstract and concrete models

• Specification for control and timing with dataflow
  – Scenario aware, heterochronous, core-functional dataflow
  – Parameterized models

• Other hardware specific problems
  – Behavioral interface formalisms (IP-XACT)
  – IP interface standardization
Y-Chart: A Disciplined System Design Methodology

Application Model (and Constraints) -> Analysis and Mapping -> Performance Evaluation -> Deployment

Platform Model (and Constraints) -> Analysis and Mapping

- Representative formal models
- Efficient analysis and optimization
- Fast and accurate simulation
- Reliable verification


An Open API and Software Stack

• Create products for new problem spaces
• Leverage common infrastructure across products
• Scale implementations across platforms – desktop OSs to web
• Improve developer efficiency in creating products
• Expect infrastructure to external partners to create products
Web LabVIEW Development Process

1. Develop the thin client using Web LabVIEW on ni.com

2. Deploy the thin client to the Real-Time device

3. Run the thin client by navigating to it in a browser