From Streaming Models to Hardware and Software Implementations

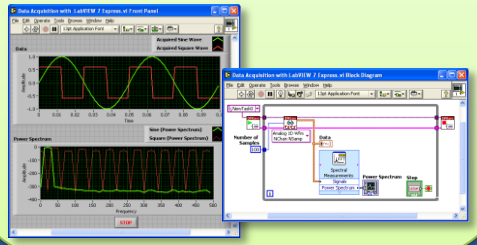
Kaushik Ravindran and Hugo Andrade
National Instruments Corporation, Berkeley, CA, USA

Workshop on Software Synthesis
October 14, 2011

National Instruments: What We Do

Low-Cost Modular Measurement and Control Hardware

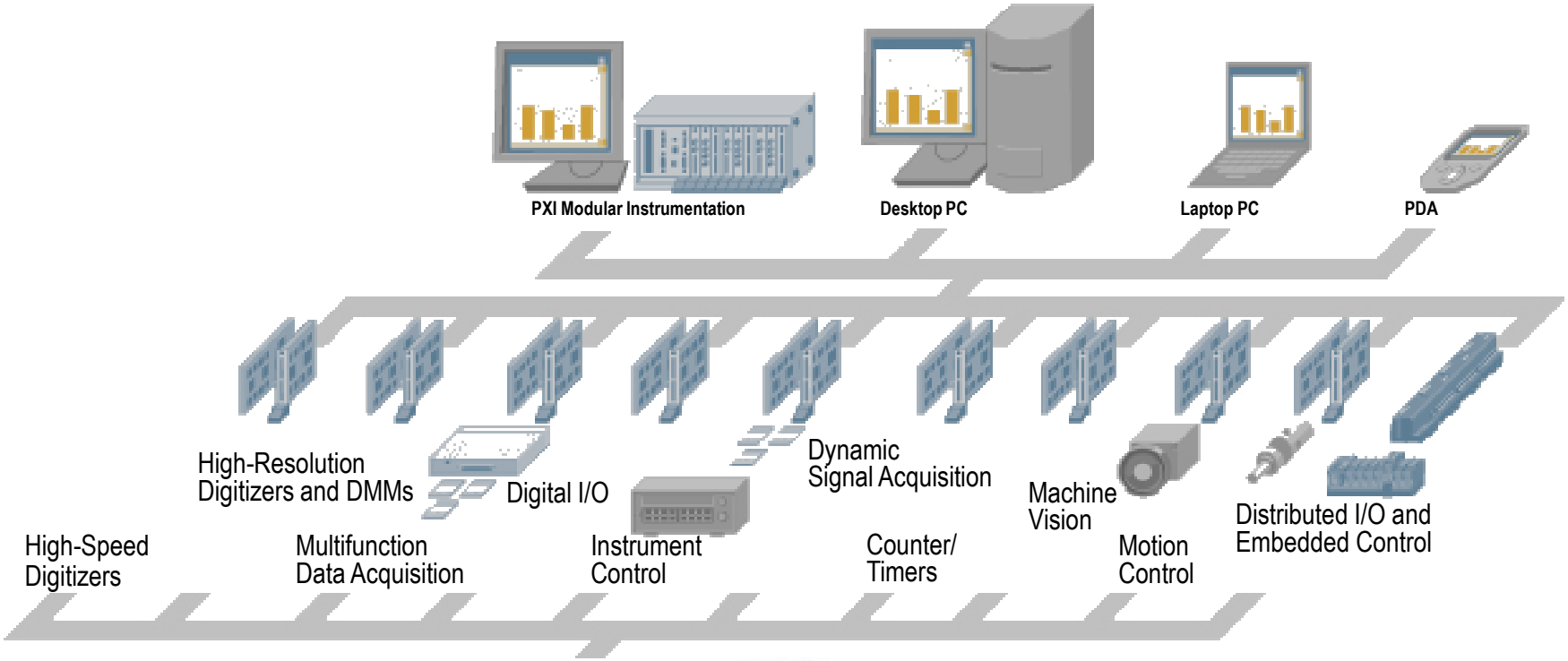
Productive Software Development Tools

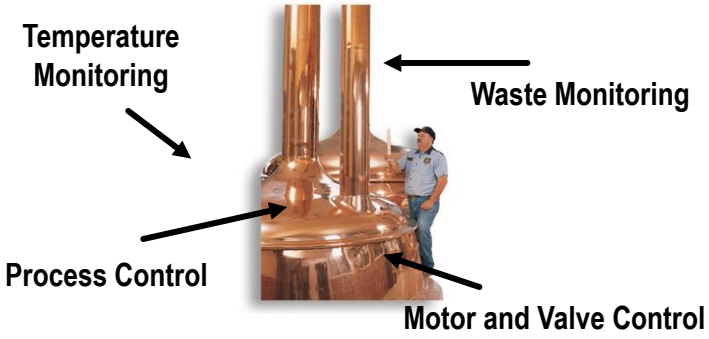


Highly Integrated Systems Platforms

Used By Engineers and Scientists for Test, Design and Control

Virtual Instrumentation

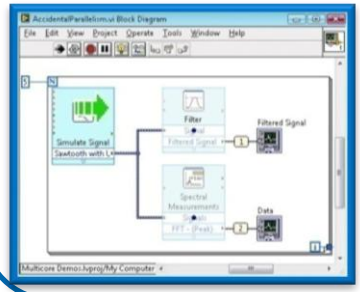




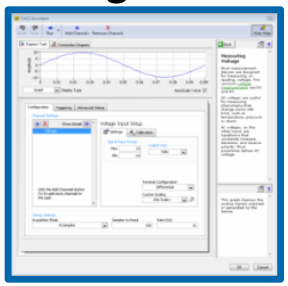
LabVIEW: Graphical System Design

High-Level Design Models

Dataflow



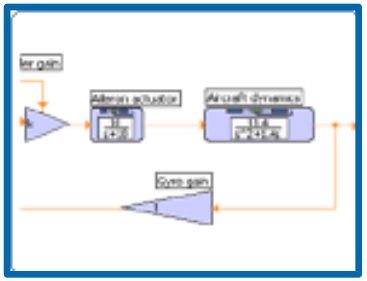
Configuration



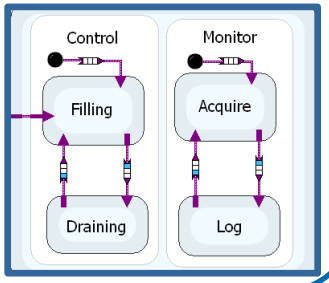
Textual Math

```
1 c = 0.285 + 0.013i;
2 [X Y] = meshgrid(x, y);
3 z = X + i*Y;
4 for k=1:30
5     z = z.^2 + c;
6 end
```

Simulation



Statechart



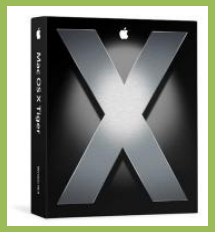
LabVIEW

Graphical Programming

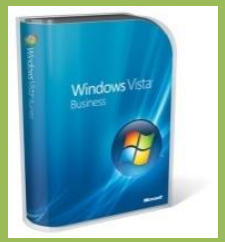
Desktop Platform

Linux

Macintosh

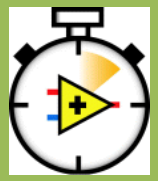


Windows

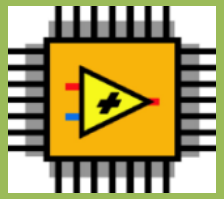


Embedded Platform

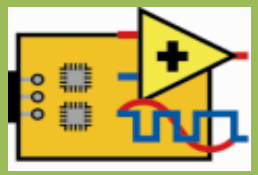
Real-Time



FPGA

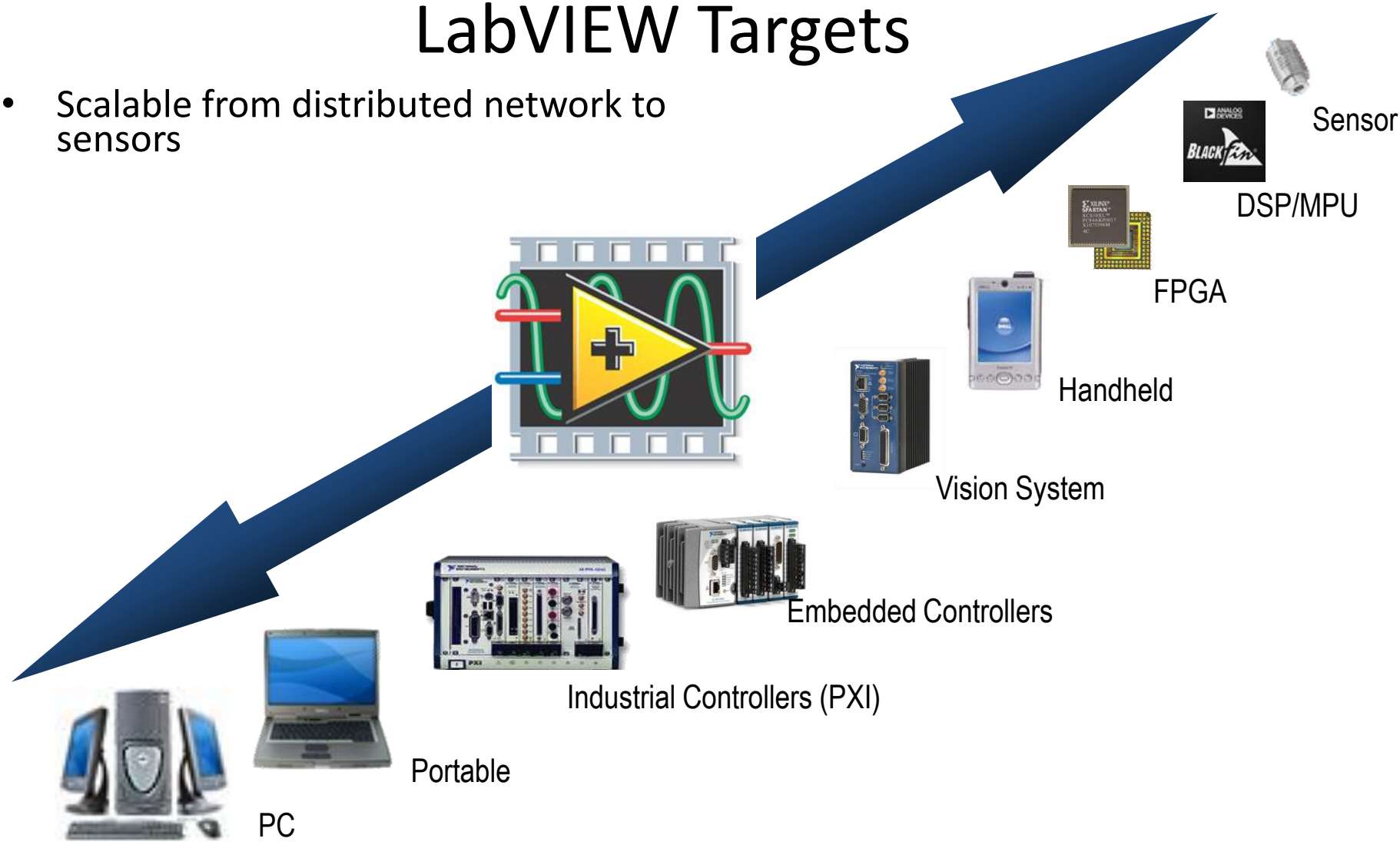


MPU



LabVIEW Targets

- Scalable from distributed network to sensors

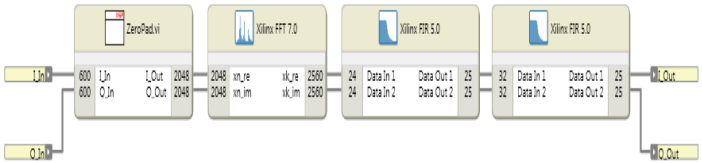


Outline

- DSP Designer framework
- Models, analysis, and exploration
- Deployment challenges
- Summary

Motivation

Concurrent Application

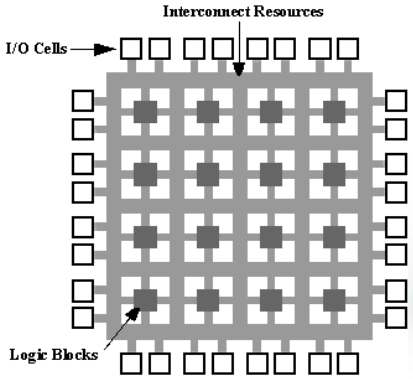


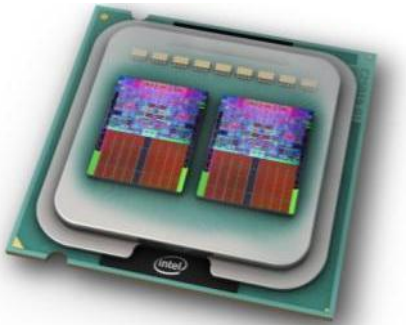
Application trends

- **1000's of parallel tasks**
- Large node/channel counts
- High performance requirements
- E.g. streaming DSP applications

Implementation Gap

Parallel Platform

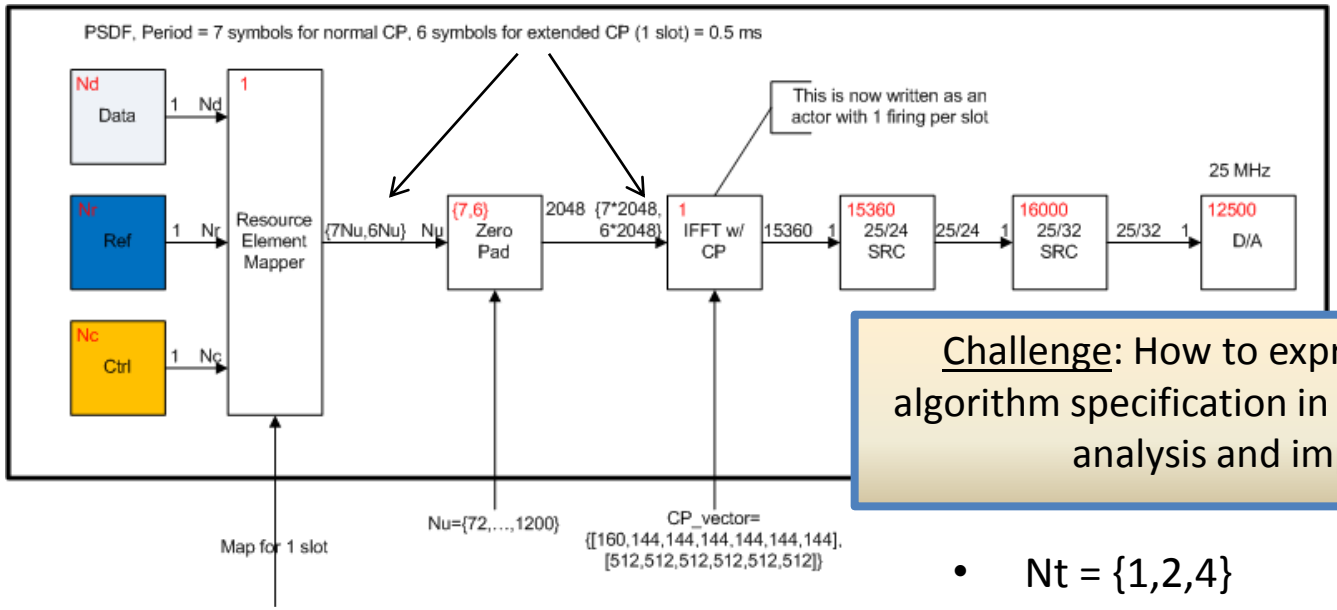




Platform trends

- **100's of processing elements**
- Heterogeneous processors and memories
- Distributed I/O
- E.g. Heterogeneous FPGA targets

Streaming Model of the OFDM Transmitter



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

- 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

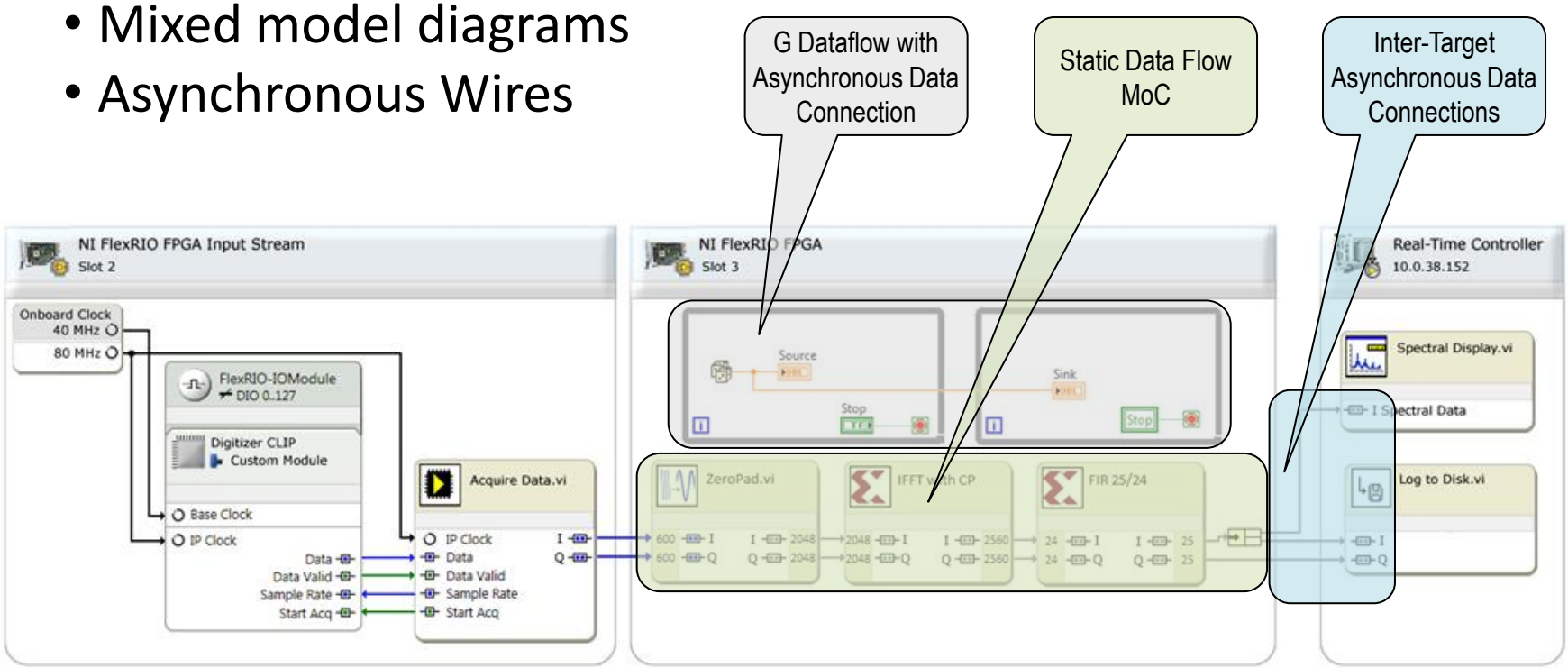
DSP Designer

- 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

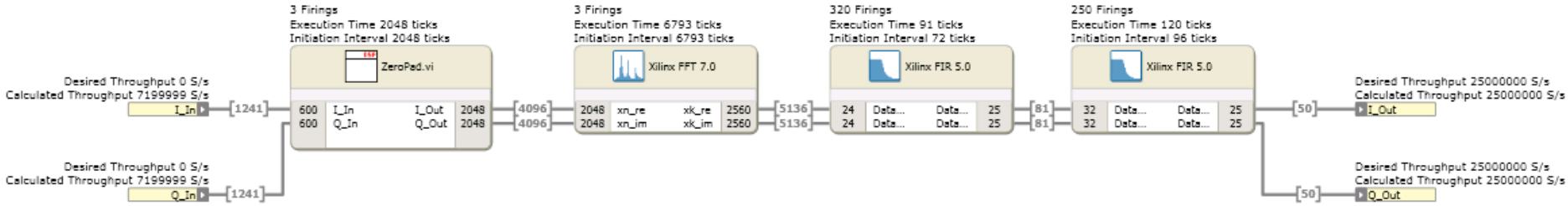
Modeling System-Level Designs

New modeling constructs

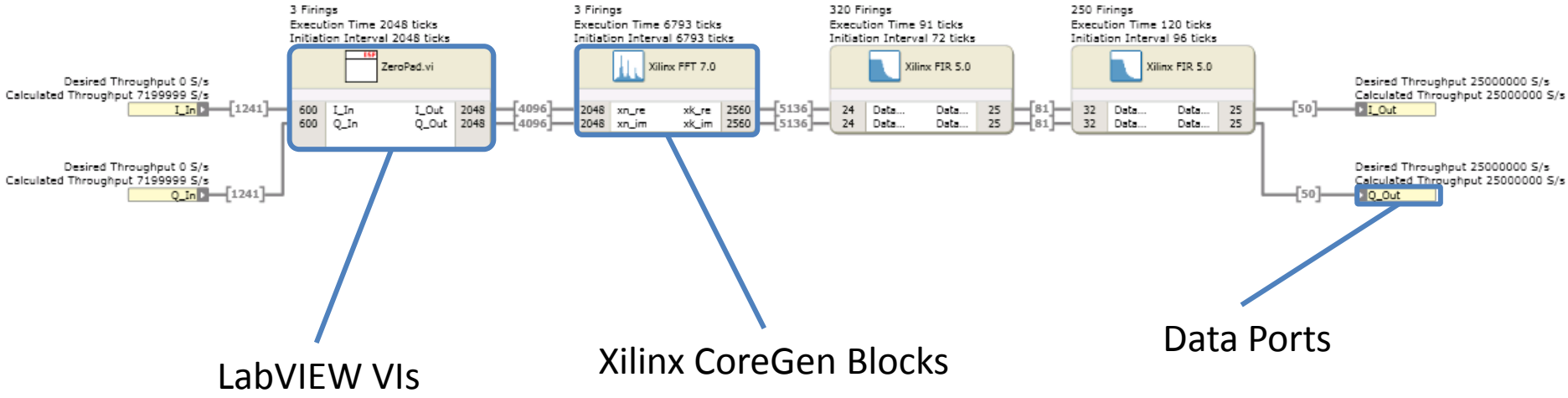
- Systems
- Targets
- Mixed model diagrams
- Asynchronous Wires



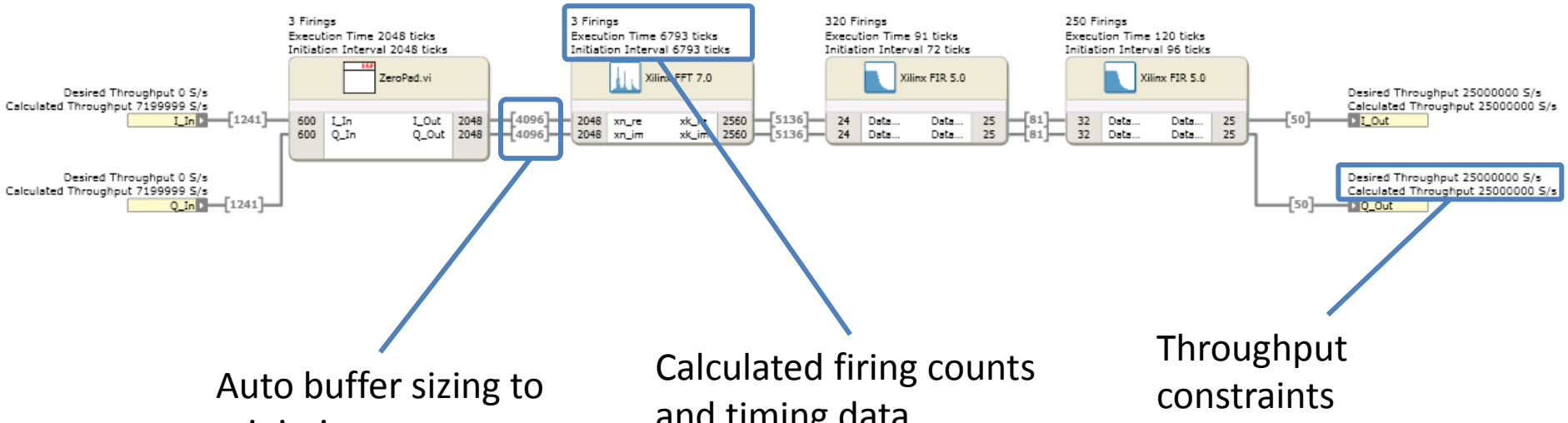
LabVIEW DSP Designer



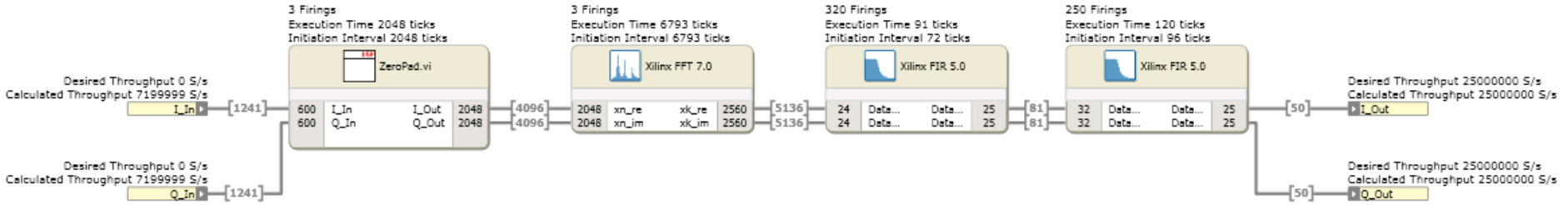
LabVIEW DSP Designer

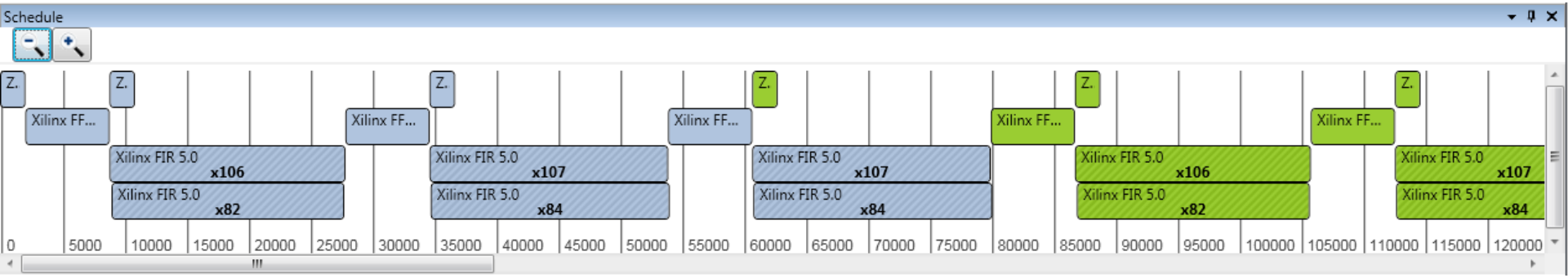


LabVIEW DSP Designer



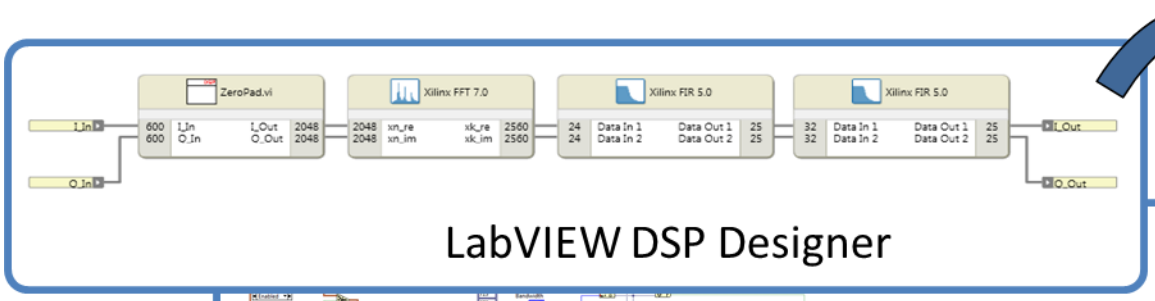
LabVIEW DSP Designer



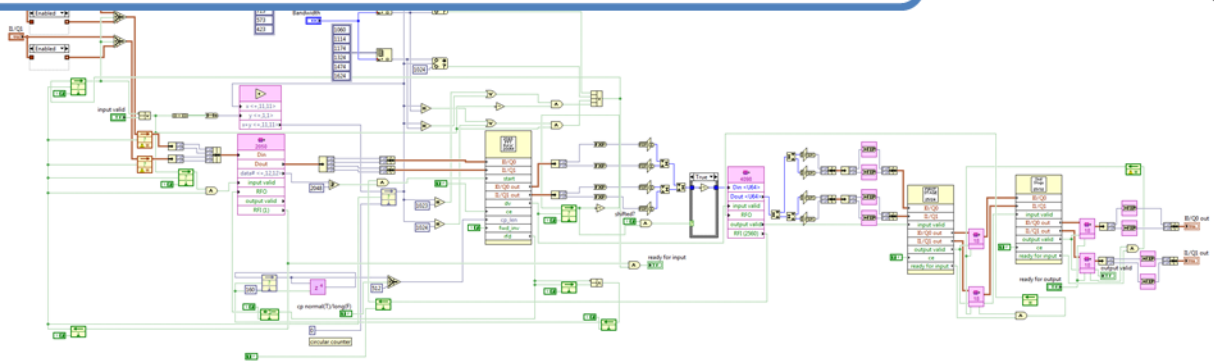


Calculated Schedule View

Value of DSP Designer



LabVIEW DSP Designer

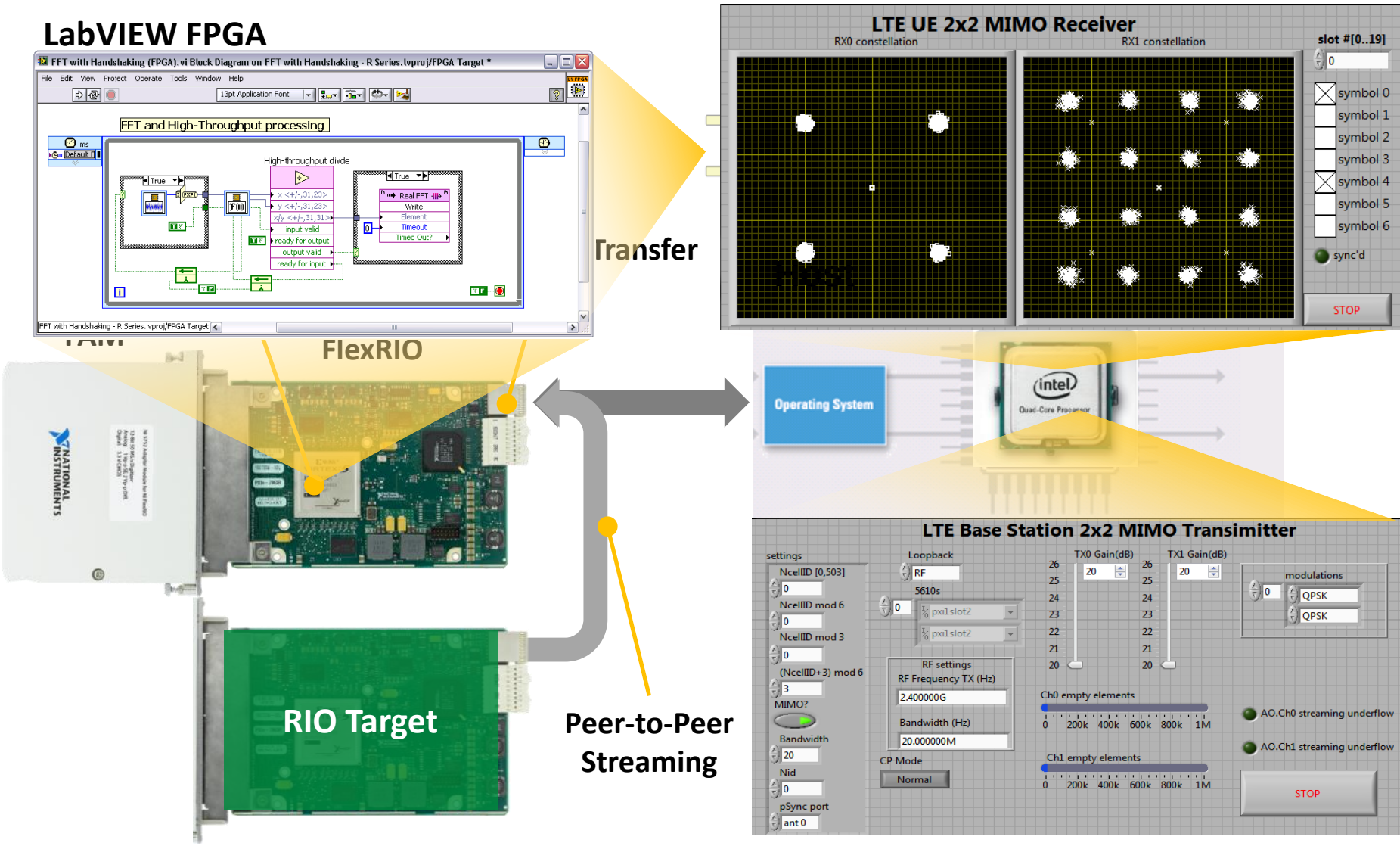


LabVIEW FPGA

```
-- synthesis translate_off                                    32_32MSPS100MHz IS
component wrapped_secondStage25o32_32MSPS100MHz
      port (
      clk: IN std_logic;
      ce: IN std_logic;
      nd: IN std_logic;
      rfd: OUT std_logic;
      rdy: OUT std_logic;
      din_1: IN std_logic_VECTOR(31 downto 0);
      din_2: IN std_logic_VECTOR(31 downto 0);
      din_3: IN std_logic_VECTOR(31 downto 0);
      din_4: IN std_logic_VECTOR(31 downto 0);
      dout_1: OUT std_logic_VECTOR(33 downto 0);
      dout_2: OUT std_logic_VECTOR(33 downto 0);
      dout_3: OUT std_logic_VECTOR(33 downto 0);
      dout_4: OUT std_logic_VECTOR(33 downto 0));
end component;
```

VHDL

RF Design Flow



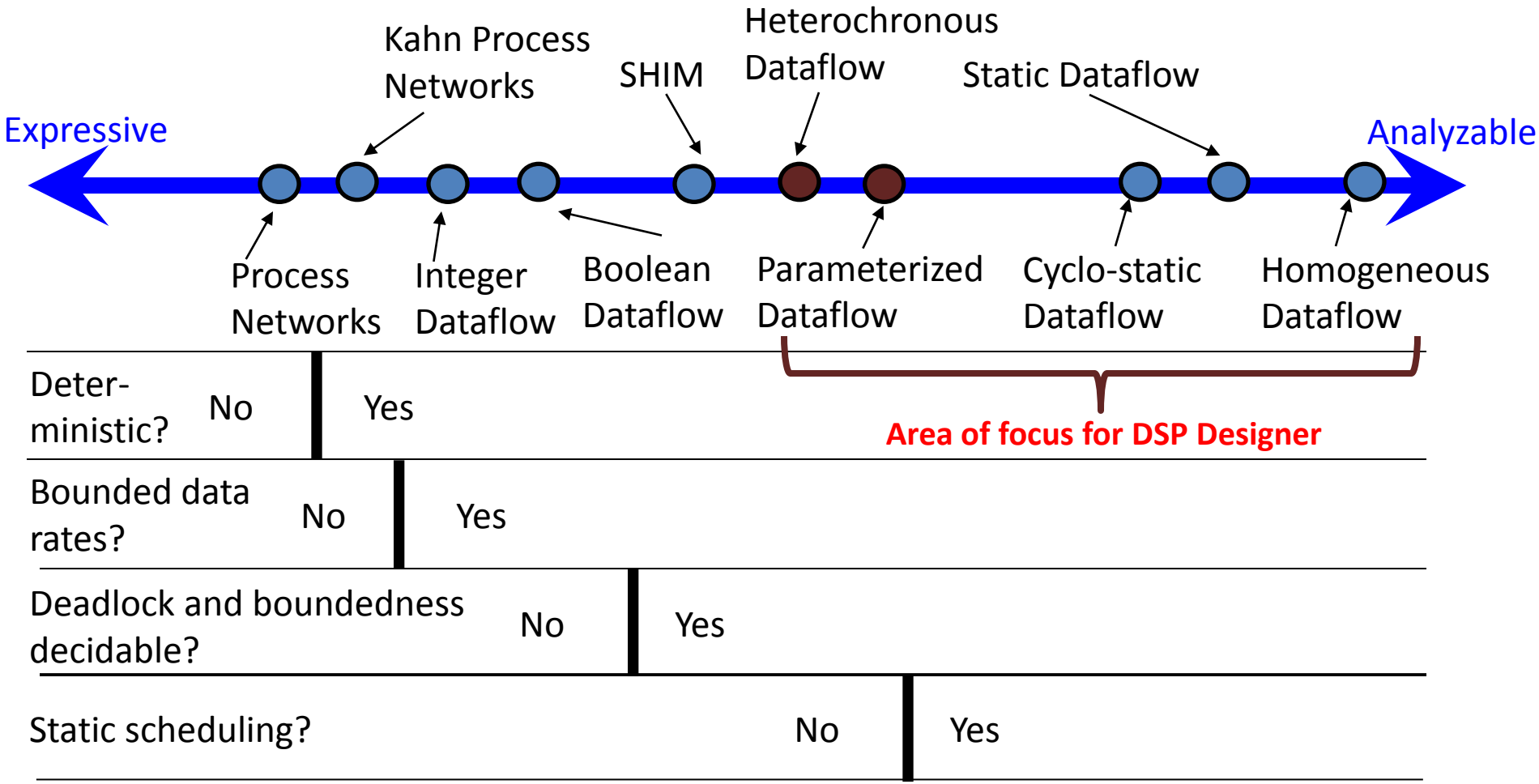
Outline

- 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



Key trade-off: Analyzability vs. Expressibility

[1] Edward A. Lee, "Concurrent Models of Computation for Heterogeneous Software", EECS 290, 2004.
[2] Stephen Edwards, "SHIM: A Deterministic Model for Heterogeneous Embedded Systems", UCB EECS Seminar, 2006.

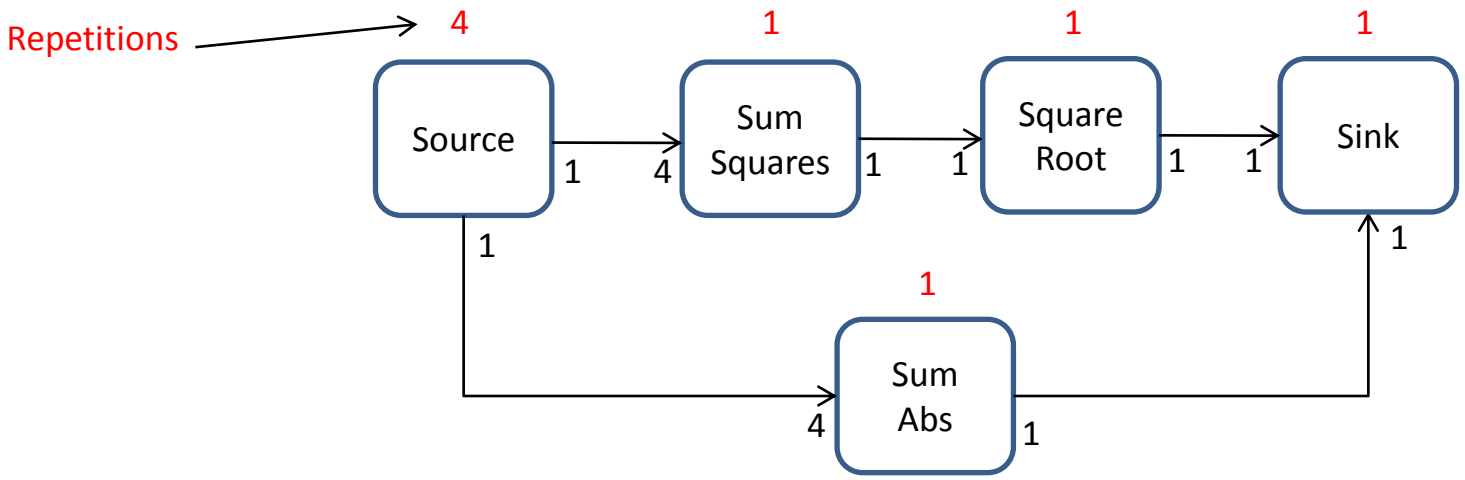
Analysis and Optimization Features

- 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

[1] S. S. Bhattacharyya, P. K. Murthy and E. A. Lee, "Software Synthesis from Dataflow Graphs," Kluwer Academic Publishers, Norwell, Mass, 1996.

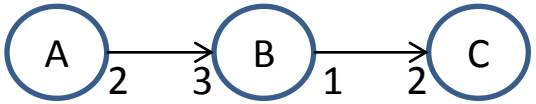
Model Validation



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

Buffer Size Optimization

Example SDF Graph

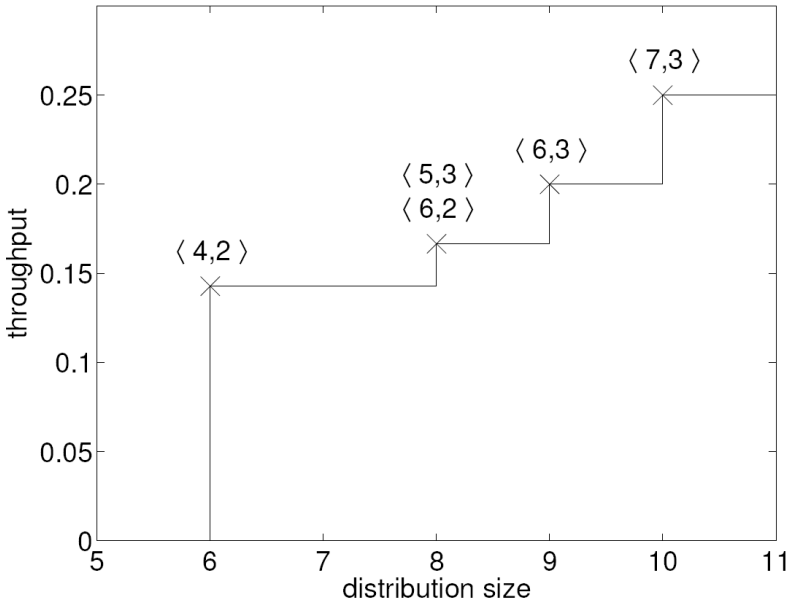


Repetitions vector: (3, 2, 1)

Problem Assumptions

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

Pareto Space of Throughput and Buffer Sizes

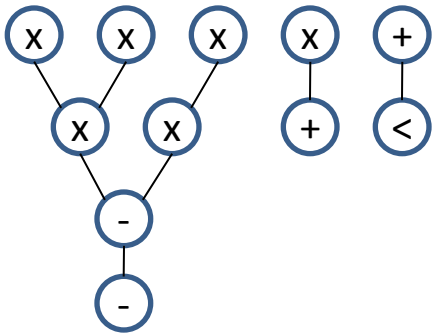


Exploration results in a Pareto space of throughput and buffer sizes

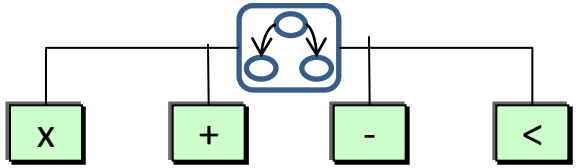
[1] S. Stuijk, M.C.W. Geilen and T. Basten, “Exploring Trade-Offs in Buffer Requirements and Throughput Constraints for Synchronous Dataflow Graphs”, DAC 2006

Resource Constrained Scheduling

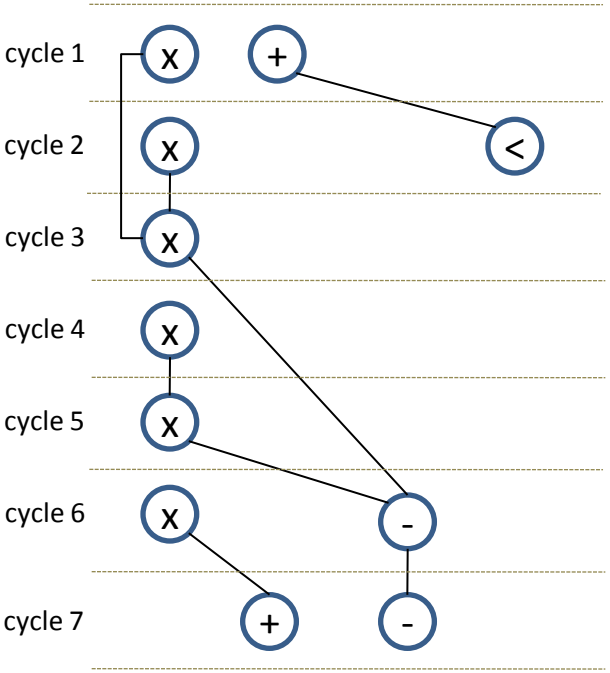
Application Model



Micro-Architecture Model



Resource Constrained Schedule



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

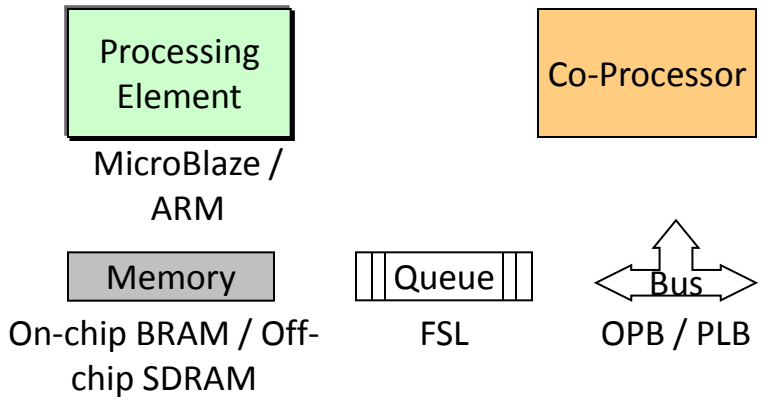
Ref: C-T. Hwang, J-H. Lee, Y-C. Hsu, "A Formal Approach to the Scheduling Problem in High Level Synthesis", IEEE Trans. On CAD, Vol 10, No 4, Apr 1991.

FPGA Based Soft Multiprocessor Systems

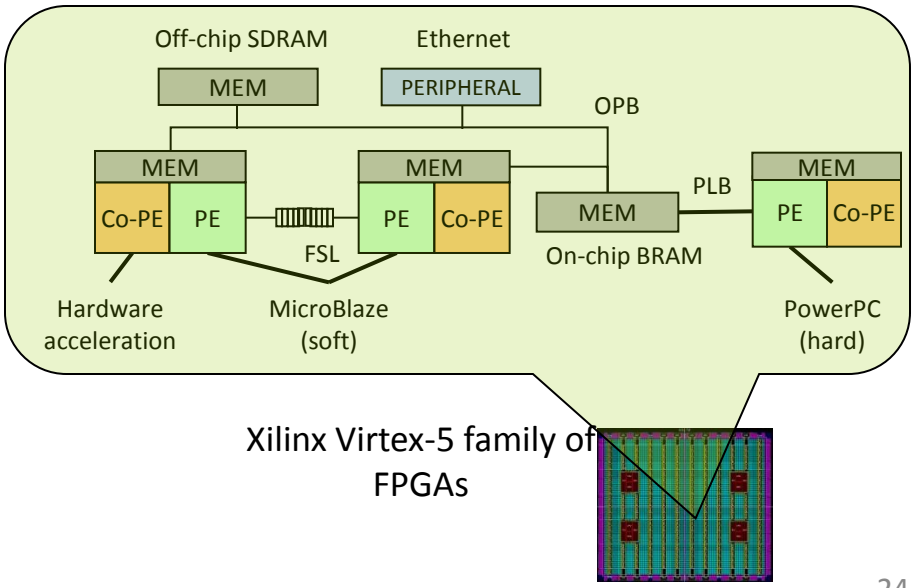
- Designer customizes multiprocessor architecture
  - Number of processors
  - Interconnection network
  - Custom co-processors

- Advantages
  - FPGA becomes accessible to software developers
  - Software compilation faster than hardware synthesis

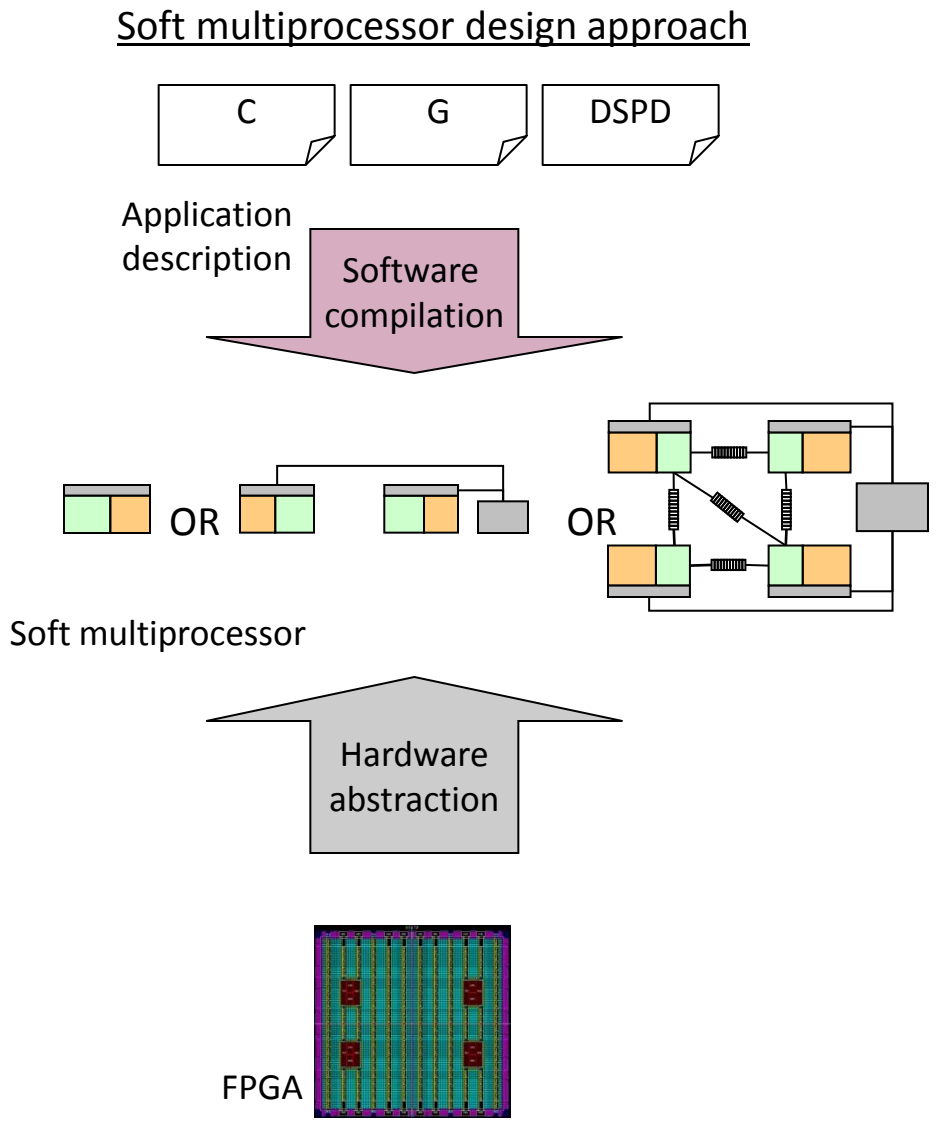
Architecture Building Blocks for Xilinx FPGAs



Multiprocessor Configuration



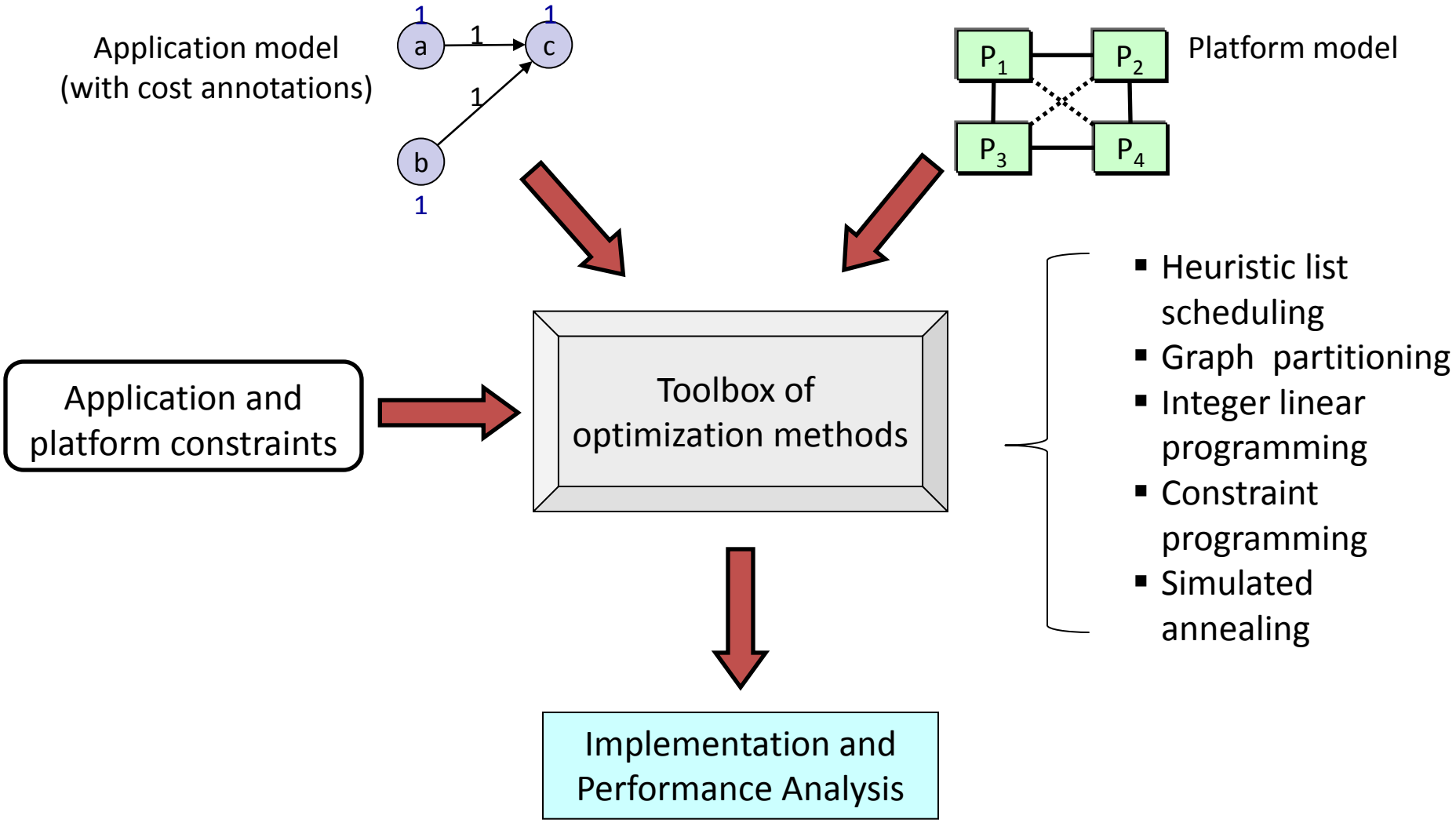
Designing Soft Multiprocessor Systems on FPGAs



- 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

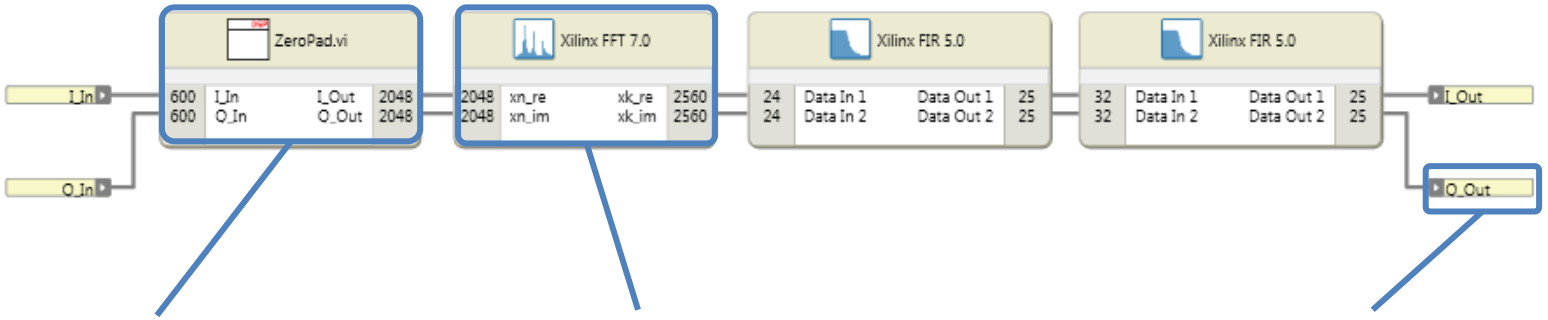


Outline

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

IP Integration

Streaming Model of the OFDMA Transmitter



User HDL

Hardware IP

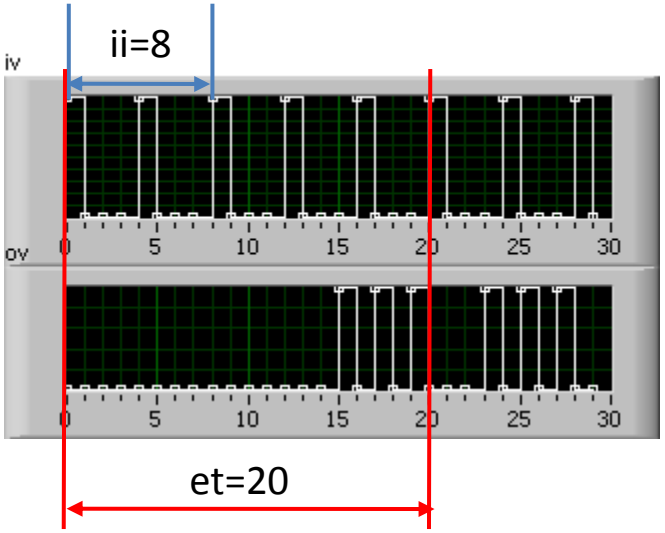
I/O with throughput requirements

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

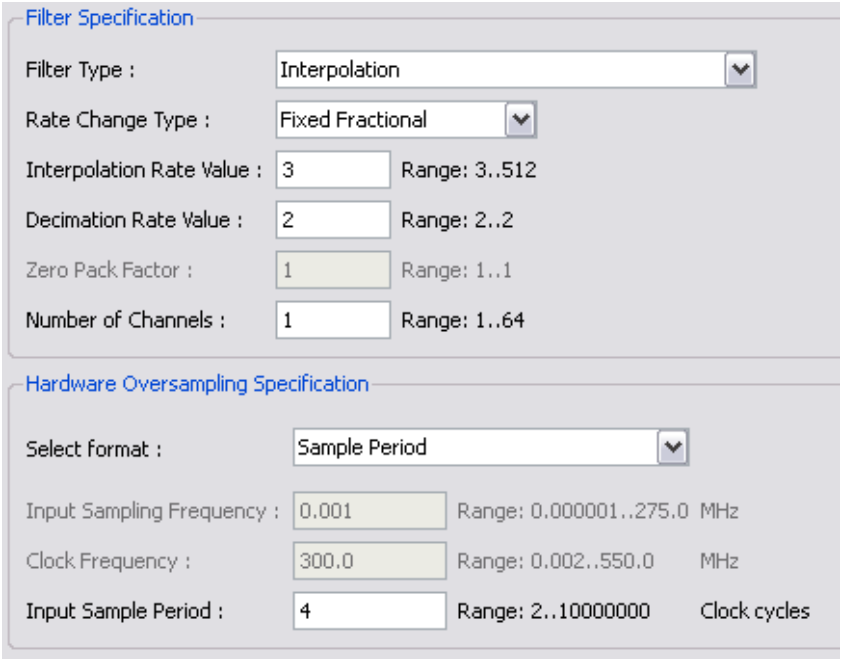
Timing Models for IP Blocks

Rational Resampler

Execution time: 20
Initiation interval: 8

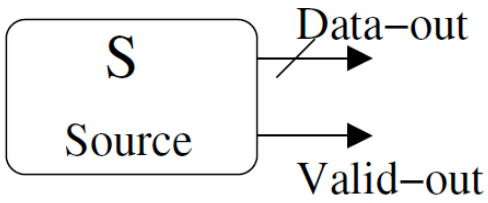


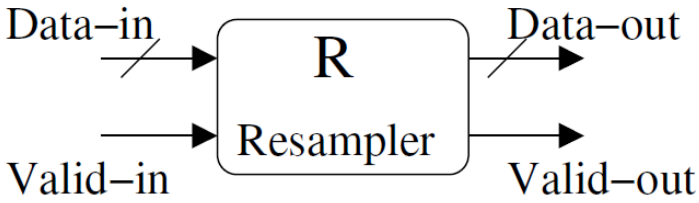
- Execution time: time to complete one firing, i.e. read 2 samples and produce 3 samples
- Initiation interval: minimum time between start of successive firings



[1] Xilinx Inc., Datasheet for CoreGen Resampler block.

Example: Source-Resampler System

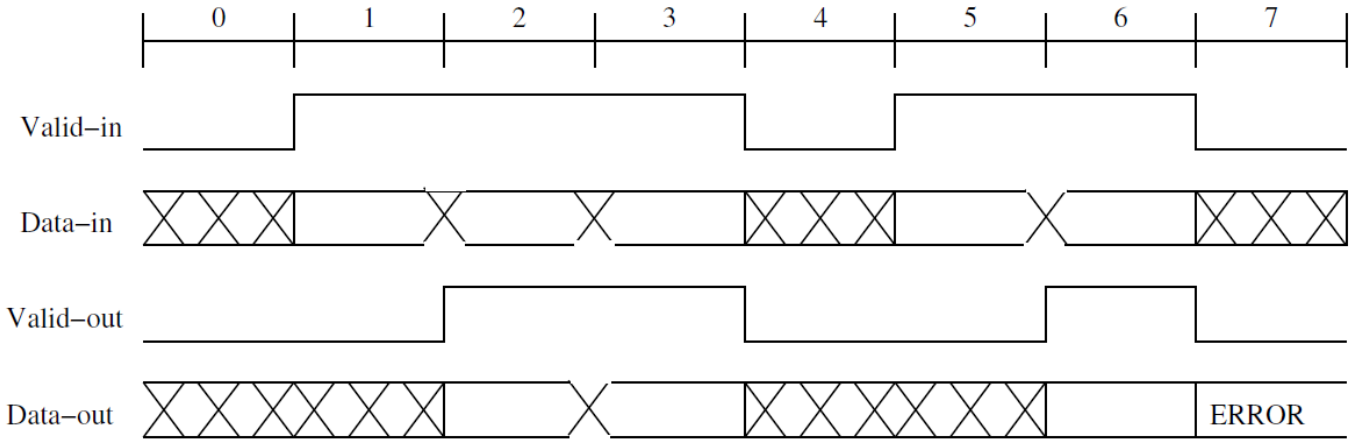




S outputs 1 sample every 2 cycles

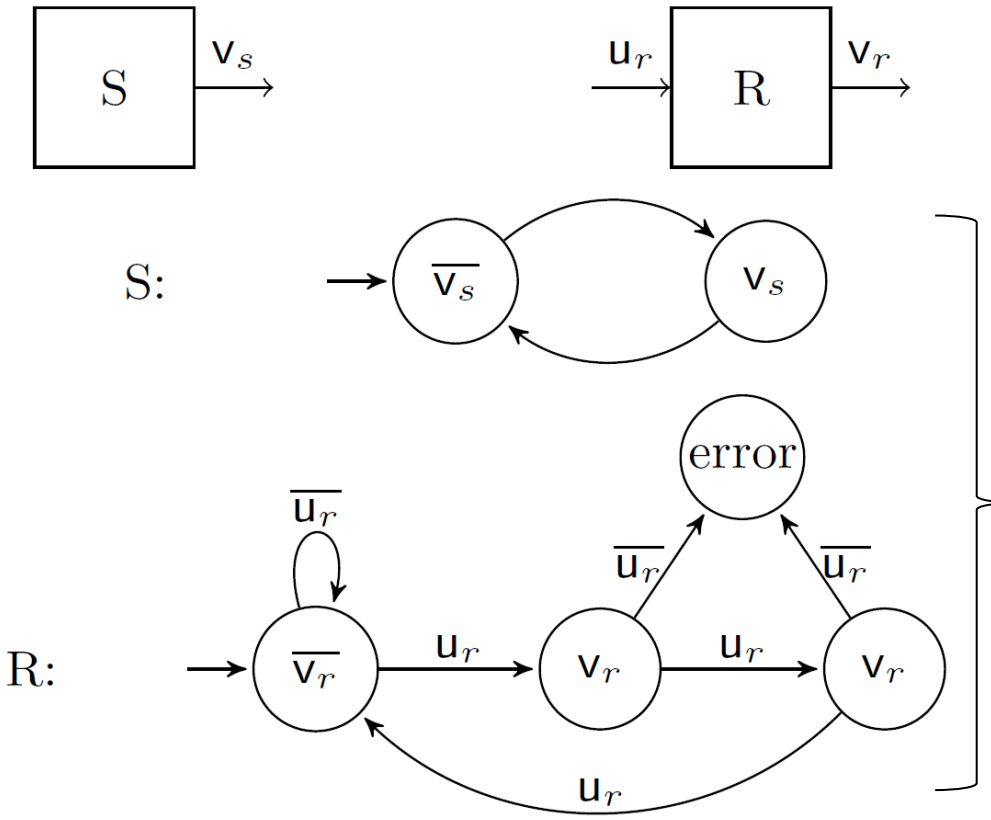
R reads 3 inputs samples in consecutive cycles and outputs 2 samples in the 2nd and 3rd cycles

Timing Diagram of Resampler



Objective: Create a valid composition of these blocks so that behavior and timing constraints are respected

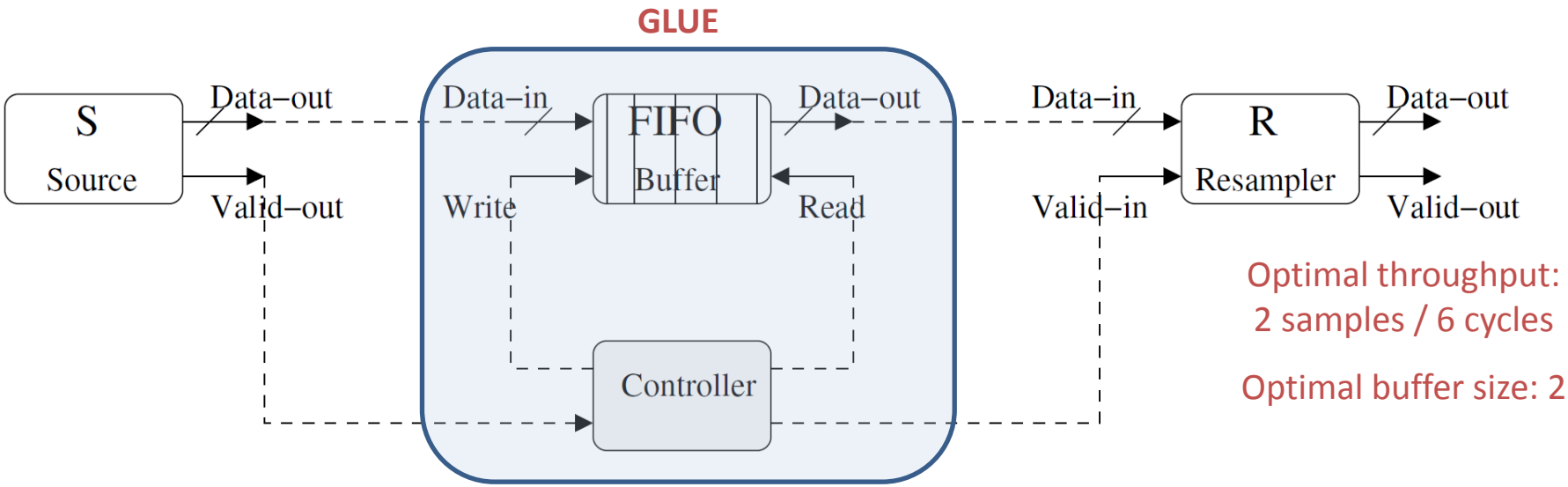
Source-Resampler: Interface Signals



FSM models for S and R

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

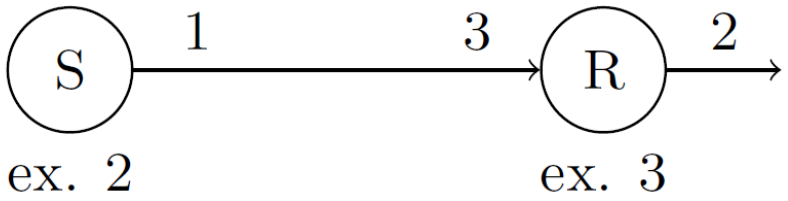
Solutions to the Glue Design Problem

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

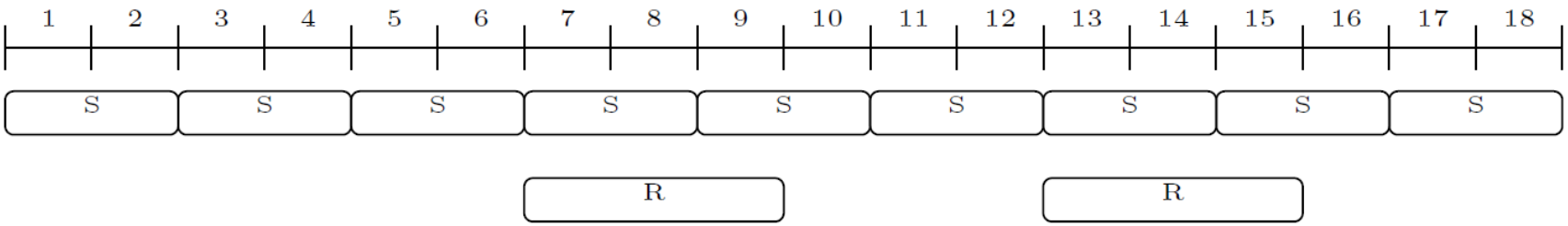
- Solutions based on abstract models
  - Enable efficient static analysis
  - Support automated synthesis

- 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)

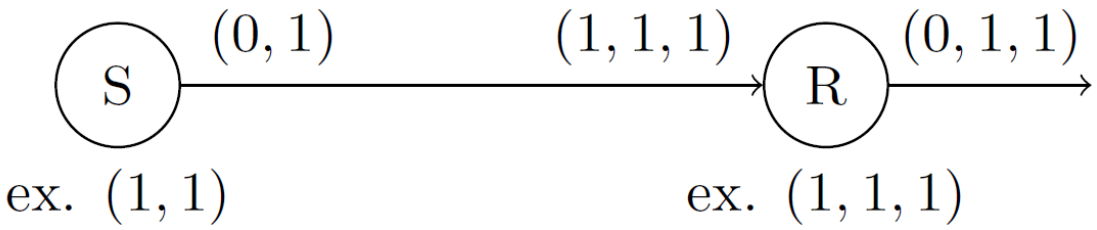


Limitations of the SDF model:

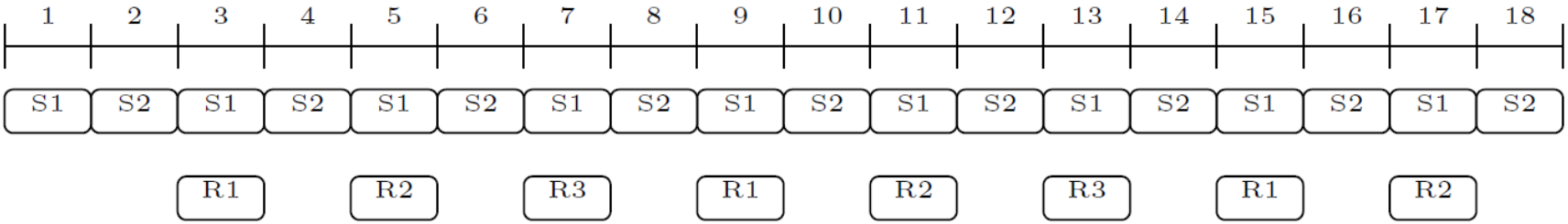
- Does not capture how tokens are accessed
- Analysis conservatively allocates space for tokens from firings of S that occur while R executes
- Resulting implementation is defensive

[1] E. A. Lee and D. Messerschmitt, "Synchronous Dataflow", in Proc. of the IEEE, 75(9), 1987.
[2] S. Stuijk, M.C.W. Geilen and T. Basten, "Exploring Trade-Offs in Buffer Requirements and Throughput Constraints for Synchronous Dataflow Graphs", DAC 2006

Cyclo-Static Dataflow Model



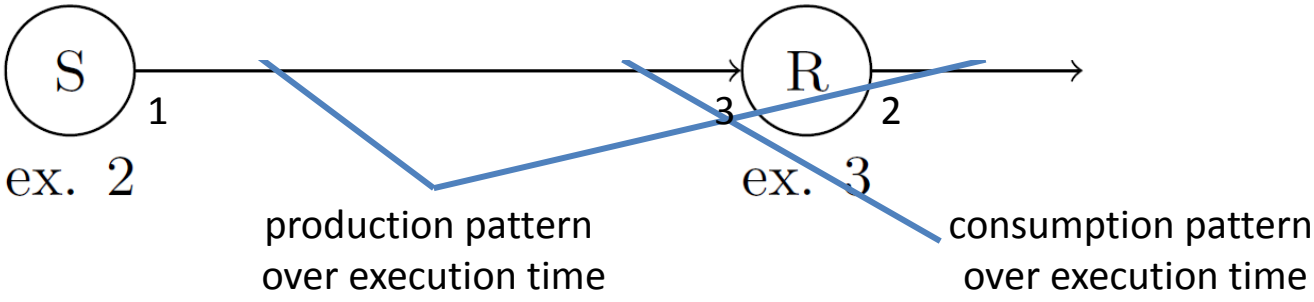
Self-timed schedule to achieve optimal throughput
(requires buffer of size 1)



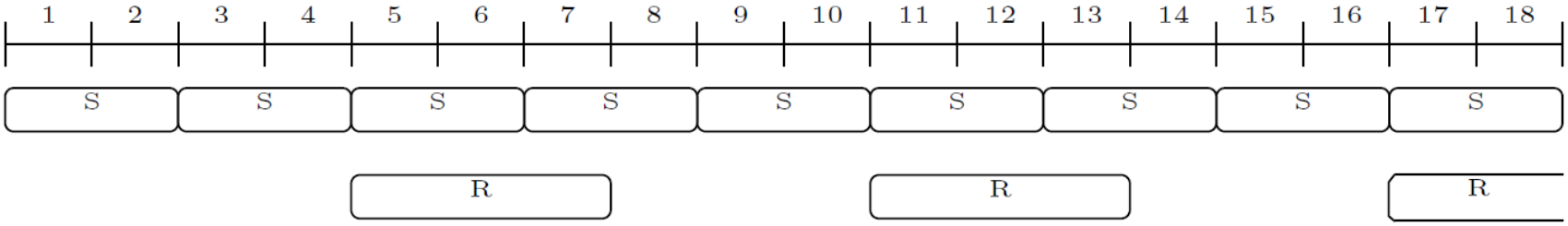
Limitations of the CSDF model:

- 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



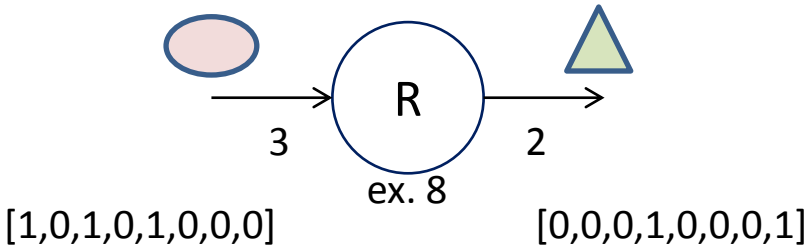
Self-timed schedule to achieve optimal throughput
(requires buffer of size 2)

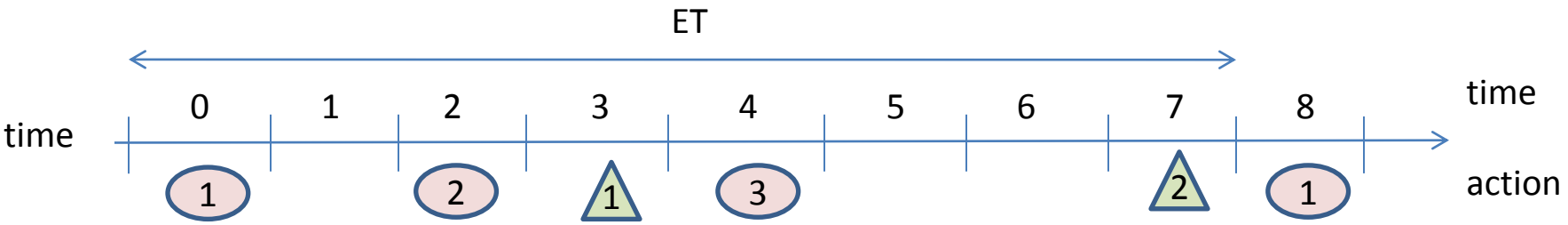


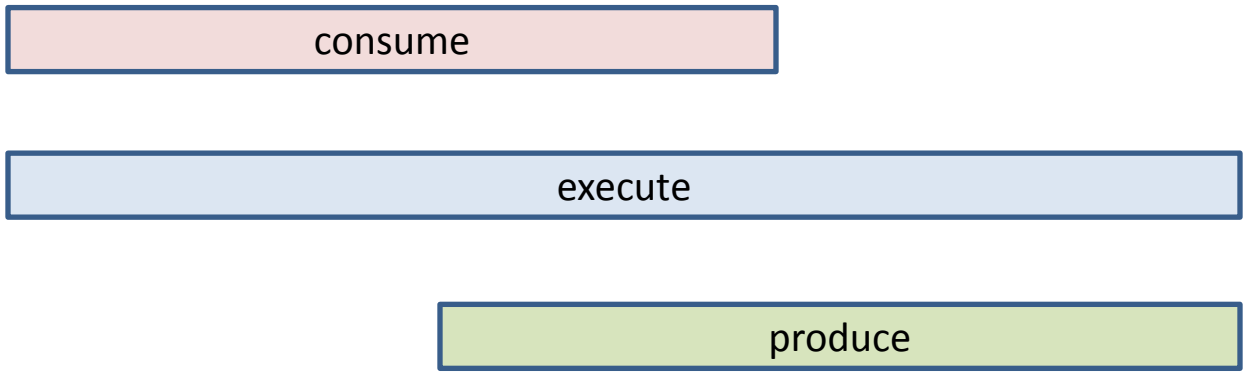
Strengths of the SDF-AP model:

- Explicitly specifies how tokens are consumed and produced in time
- Analysis yields a buffer of size 2
- Resulting implementation is correct and non-defensive

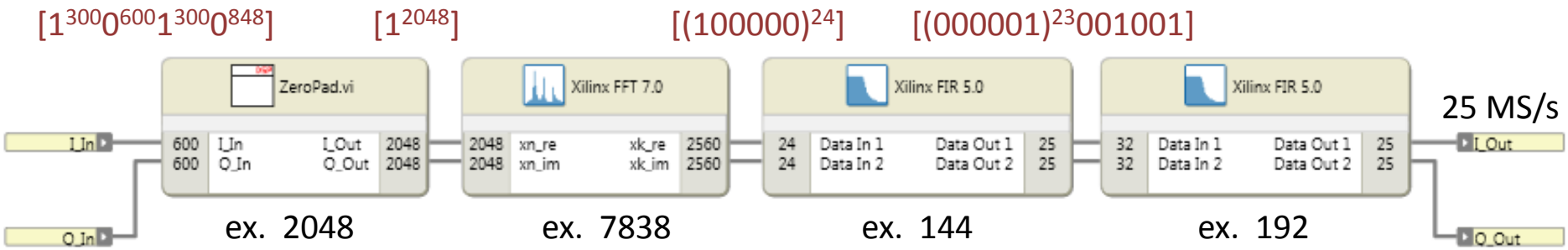
Access Pattern Example







Case Study: OFDMA Transmitter



Sizes

SDF	600	2048	2133	56	25
SDF-AP	600	1	2133	2	25

- 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

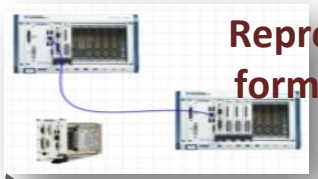
Directions Ahead

- 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

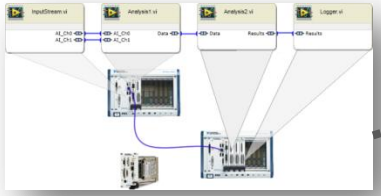
Application Model (and Constraints)

Platform Model (and Constraints)



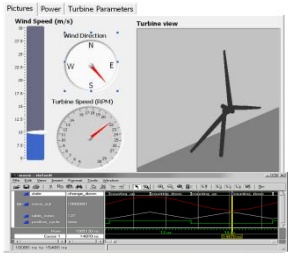
Representative formal models

Analysis and Mapping



Efficient analysis and optimization

Performance Evaluation



Fast and accurate simulation

Deployment

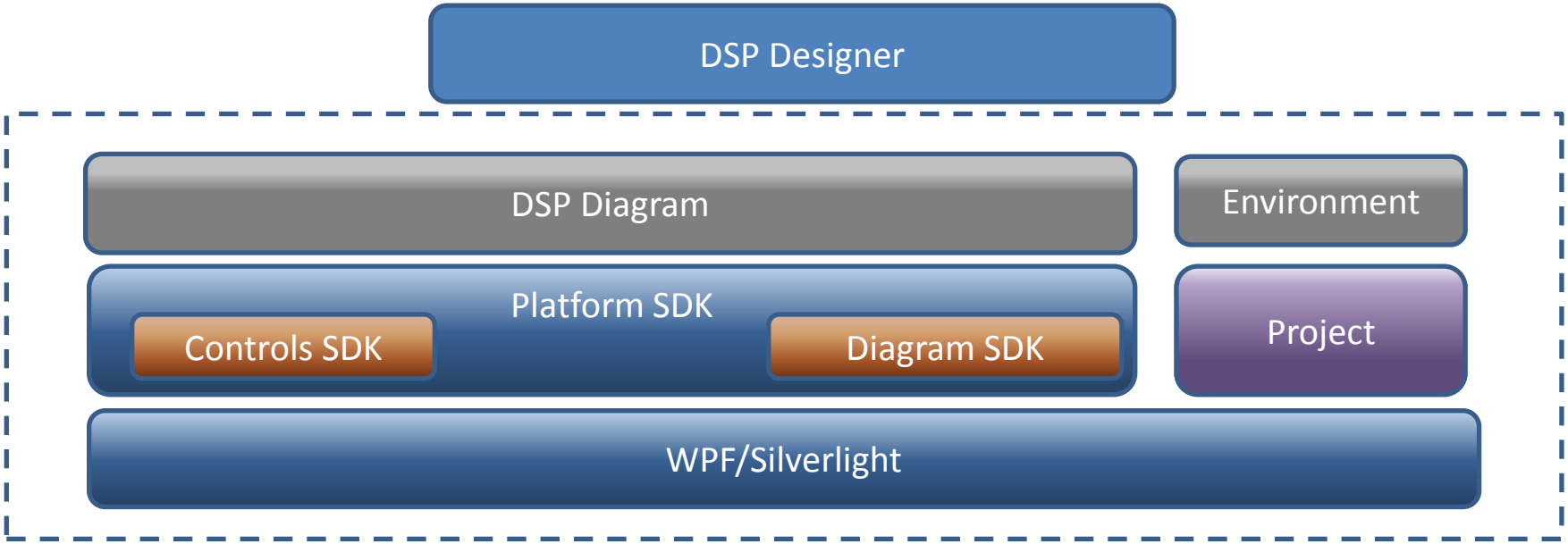
Reliable verification

[1] B. Kienhuis, E. F. Deprettere, P. Wolf, K. A. Vissers. "A Methodology to Design Programmable Embedded Systems - The Y-Chart Approach". SAMOS, p.18-37, Jan 2002.

[2] K. Keutzer, A. R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of Concerns and Platform-based Design. IEEE Trans. on CAD of ICs, 19(12): p.1523-1543, December 2000.

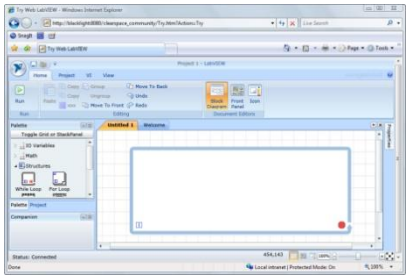
An Open API and Software Stack

- 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



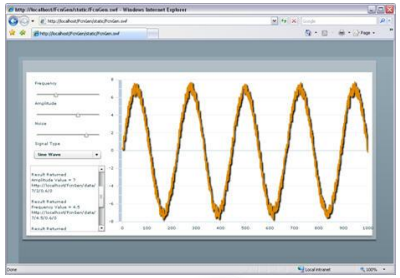
Web LabVIEW Development Process

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



NI.COM

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



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