Digital Signal Processing in LabVIEW FPGA with NI FlexRIO

Alain Moriat

NI-Denmark R&D

Agenda

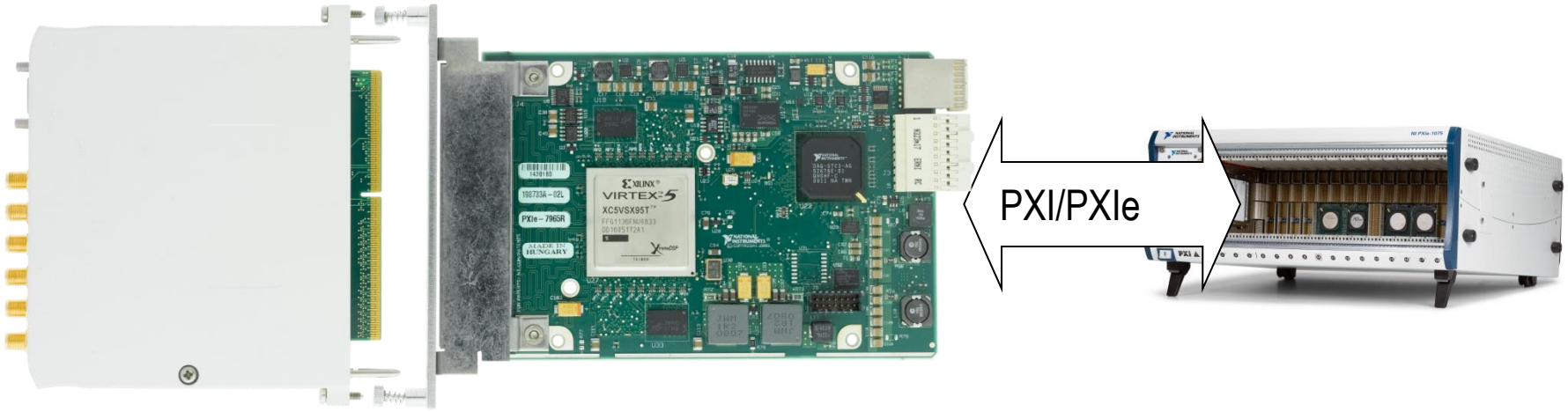
- Introduction to FlexRIO and LabVIEW FPGA Technology
- Fixed-Point Data Type
- High Throughput Design Process
- Case Study: FIR Filter
- Some Demos…

Questions to the Audience

- I attended Eric's FPGA presentation this morning
- I have never used LabVIEW FPGA
- I have used LabVIEW FPGA but I am not familiar with concepts like Fixed-point, High-throughput Maths, ...
- I am familiar with the concepts above and/or used to program with VHDL

Hardware

NI FlexRIO System



NI FlexRIO Adapter Module

- Interchangeable I/O
- Customizable by users
- NI FlexRIO Adapter Module Development Kit (MDK)

NI FlexRIO FPGA Module

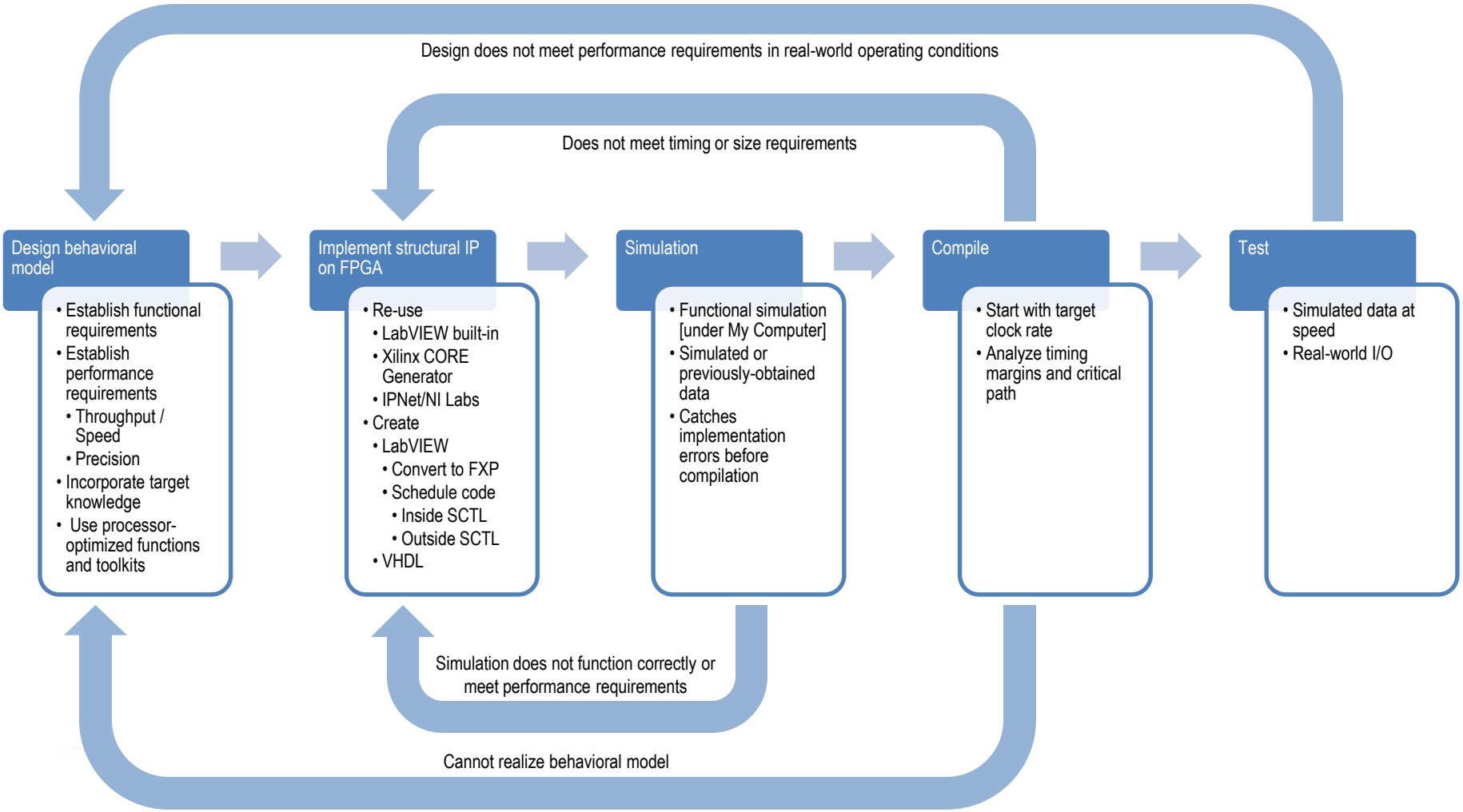
- Virtex-5 FPGA
- 132 digital I/O lines
- Up to 512 MB of DRAM

PXI Platform

- Synchronization
- Clocking/triggers
- Power/cooling
- Data streaming

HIGH THROUGHPUT NI FLEXRIO FPGA DESIGN PROCESS

FPGA Design Flow



Fixed-Point Math

- Greatly simplifies arithmetic on the FPGA
  - Simplifies computations
  - Size and speed advantages of integer math
- Must configure the appropriate range when using fixed-point math
- Limited to a maximum size of 64 bits.

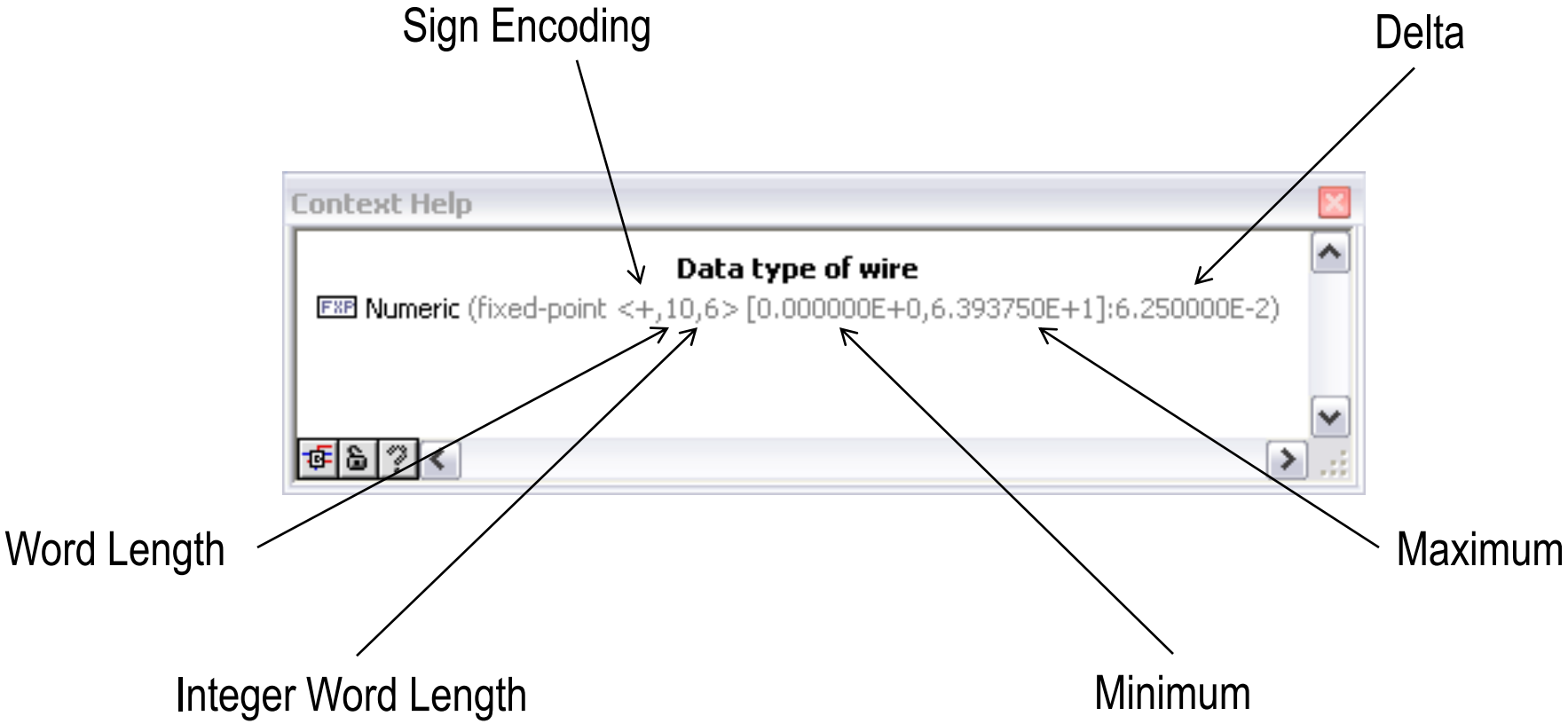
Fixed Point Terminology

Sign Encoding – The option that determines whether the fixed-point data is signed (±) or unsigned (+)

Word Length – The total number of bits used for the Fixed-Point data

Integer Word Length – The number of bits used in the integer portion of the Fixed-Point data

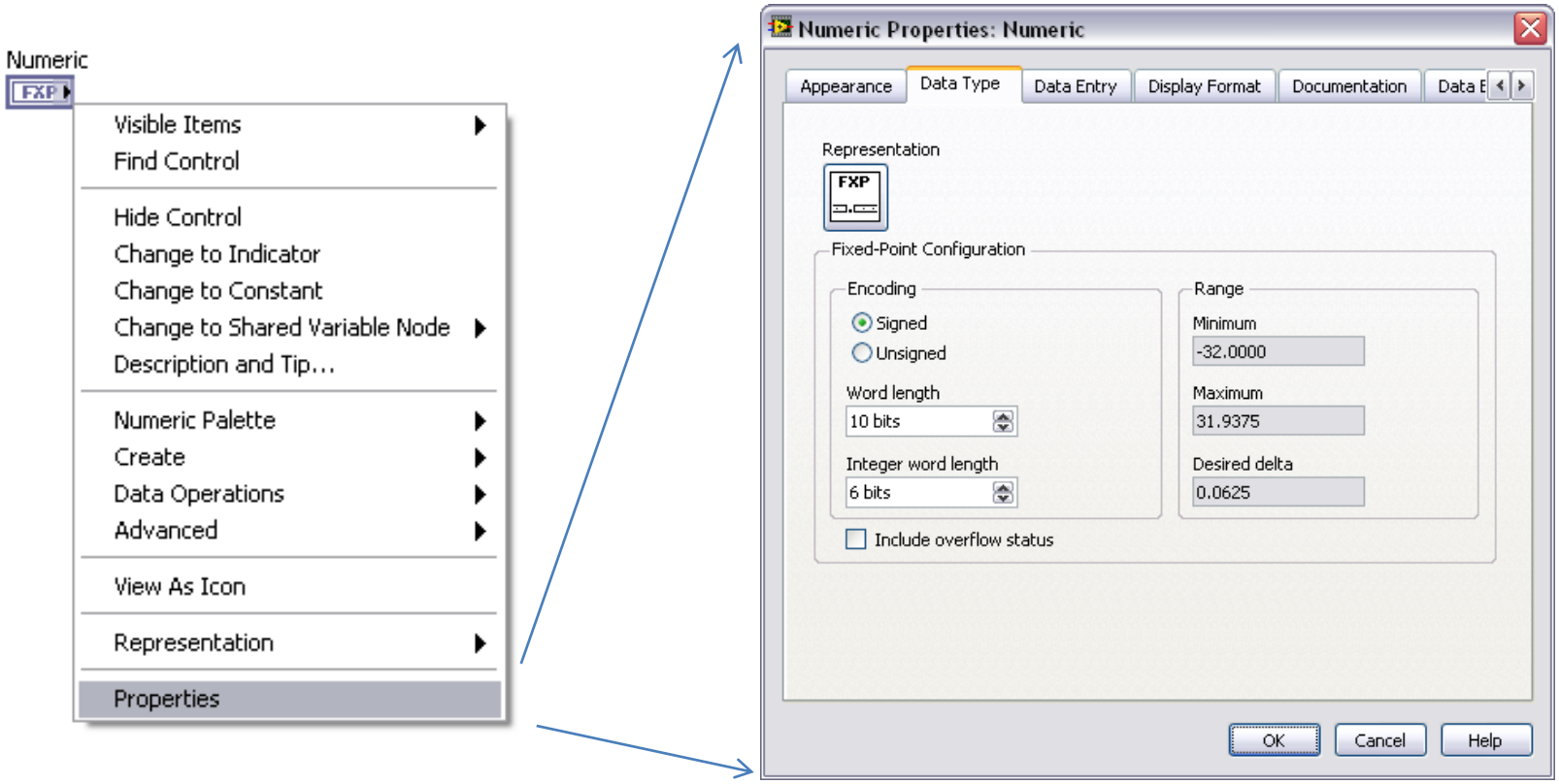
Fixed-Point Context Help



Numeric Representation Examples

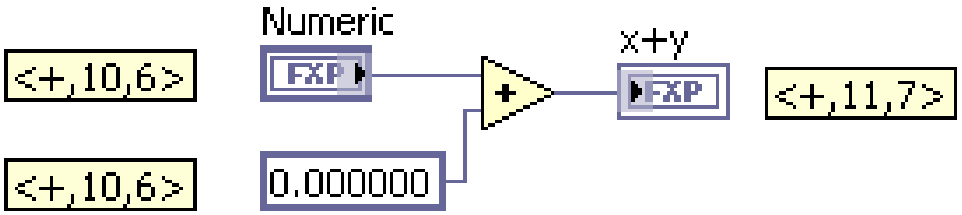
Representation	delta	Min Value	Max Value
U8 ~FXP<+,8,8>	1	0	255
I8 ~FXP <±,8,8>	1	-128	127
FXP <+,8,7>	0.5	0	127.5
FXP <+,8,6>	0.25	0	63.75
FXP <±,8,7>	0.5	-64	63.5
FXP <±,8,6>	0.25	-32	31.75
FXP <+,8,0>	0.0039	0	0.9961

Fixed Point Configuration



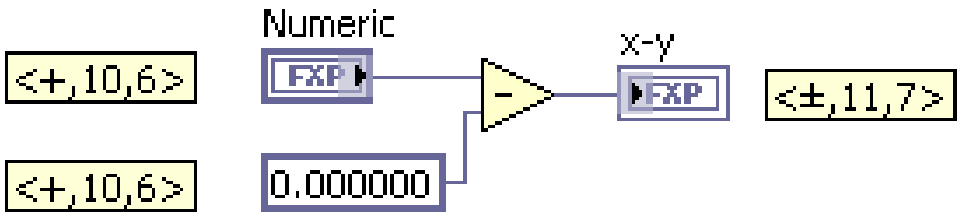
Fixed-Point Addition

- Inputs should have the same range of values
  - If inputs have different ranges, then one or both will be coerced to a range that accommodates both
- For the output, the word length and integer word length will each increase by one and the sign encoding will match the input sign encoding
  - <±,A,B> + <±,A,B> = <±,A+1,B+1>



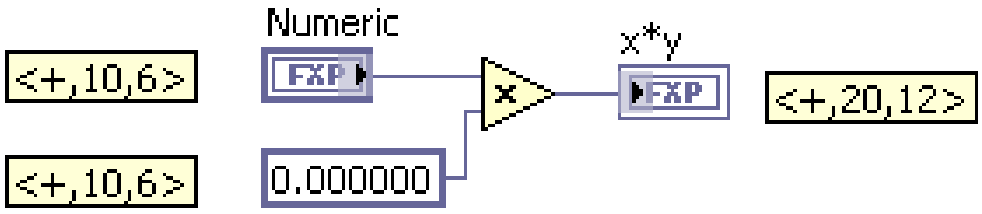
Fixed Point Subtraction

- The output representation follows the same guidelines as addition
- The output will be signed, regardless of whether or not the inputs were signed or unsigned.
  - <+,A,B> - <+,A,B> = <±,A+1,B+1>



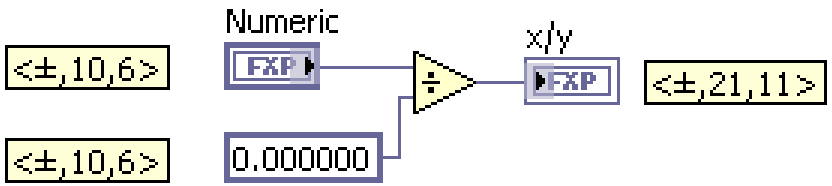
Fixed-Point Multiplication

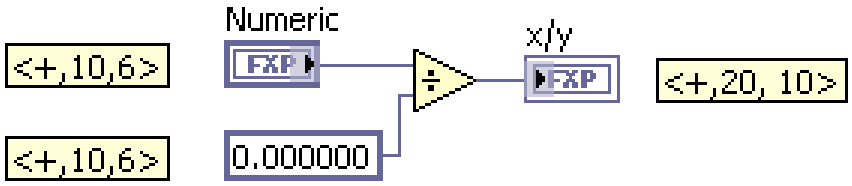
- Inputs can have different ranges of values
- For the output, the word lengths and integer word lengths are each added together
- If either input is signed, then the output will be signed. Otherwise, the output is unsigned.
  - <+,A,B> * <+,C,D> = <+,A+C, B+D>



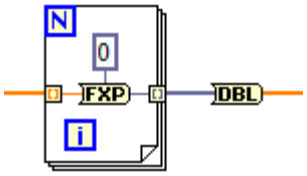
Fixed-Point Division

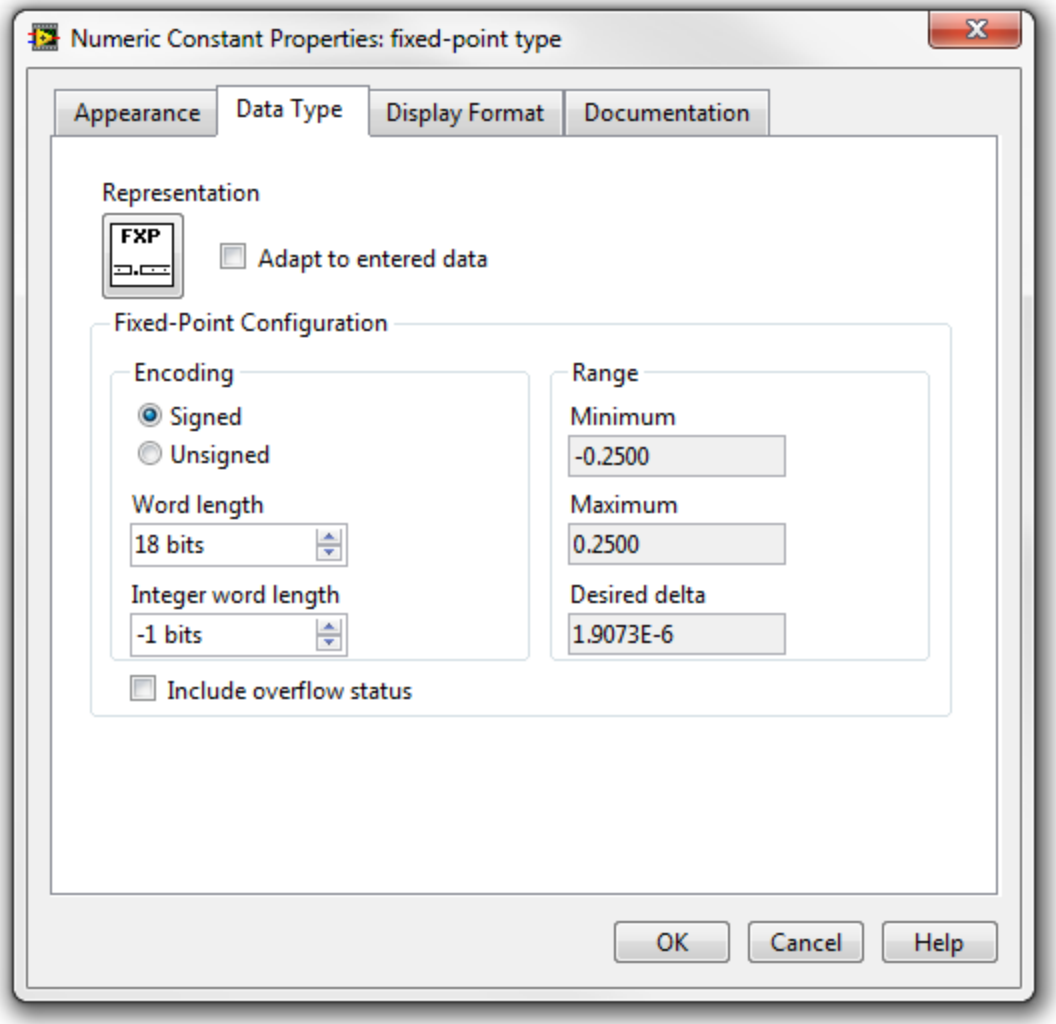
- Inputs can have different ranges of values
  - Signed and unsigned division result in different ranges of values for the output.
    - Signed: <±,A,B> / <±,C, D> = <±,A+C+1,B+C-D+1>
    - Unsigned: <+,A,B> / <+,C,D> = <+,A+C, B+C-D>





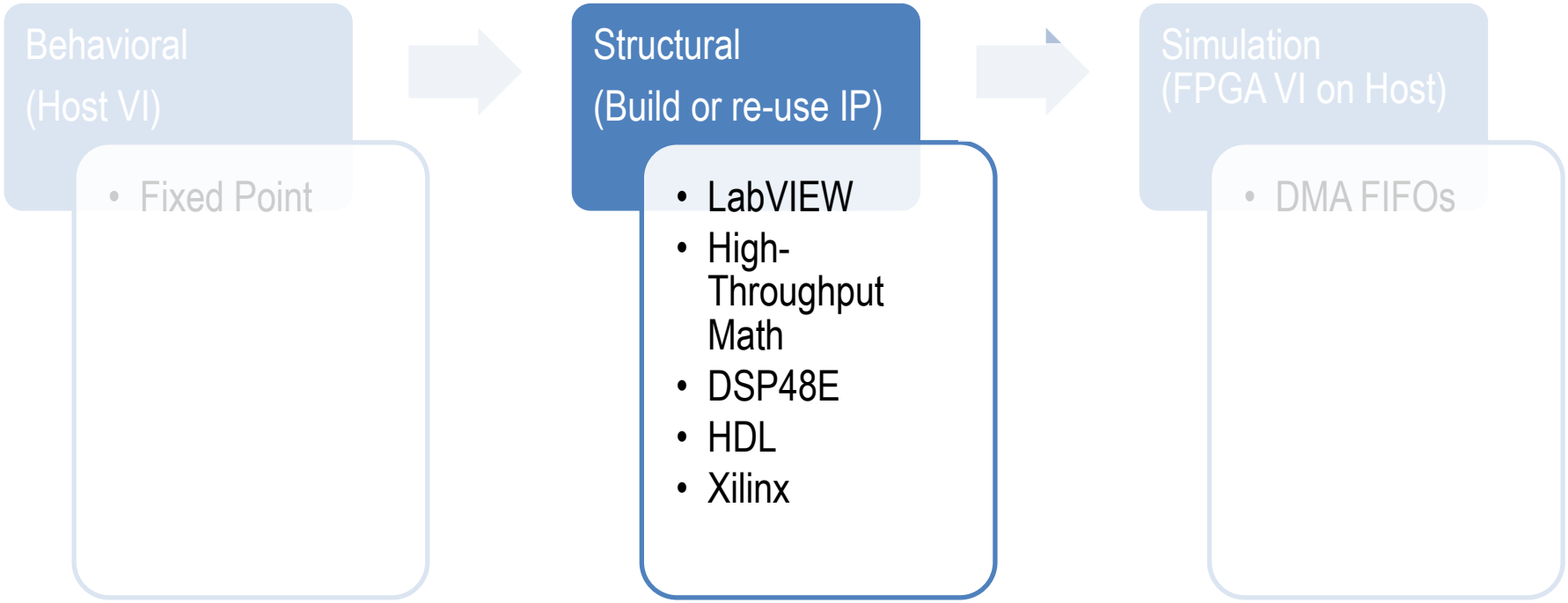
Convert to Fixed Point



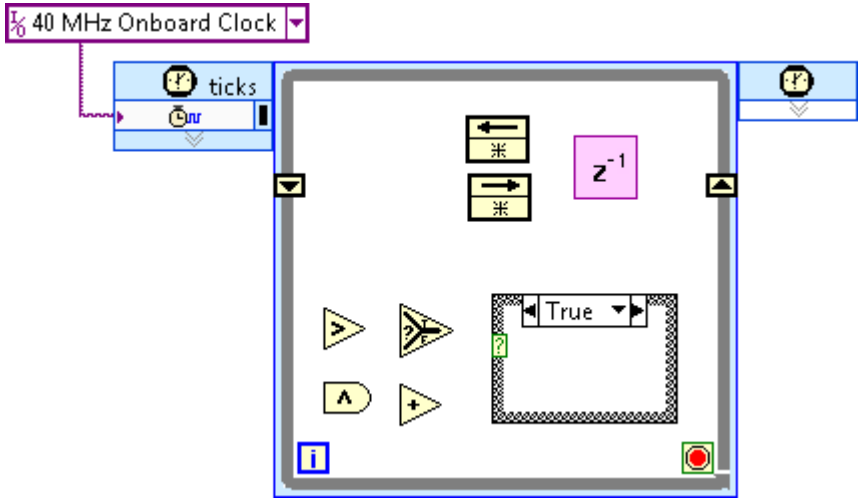


FPGA Design Flow

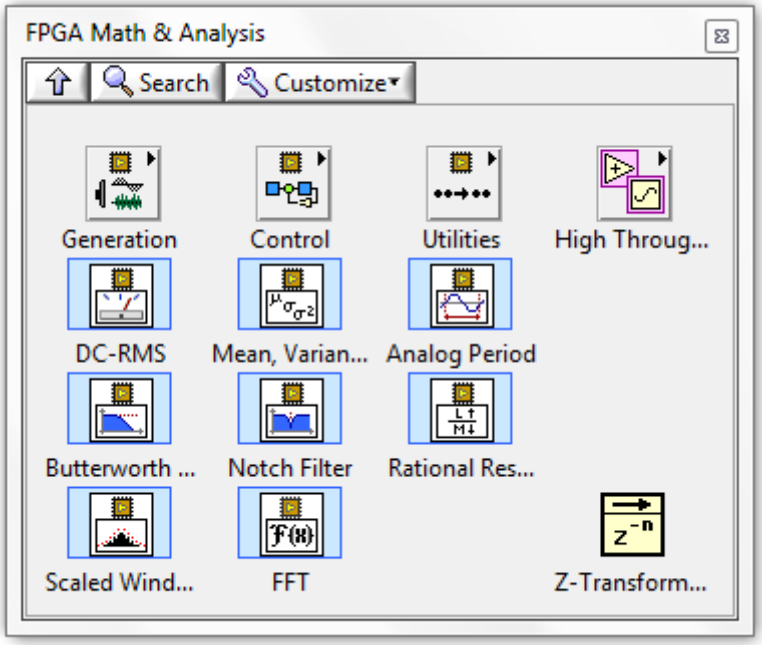
DSP Considerations

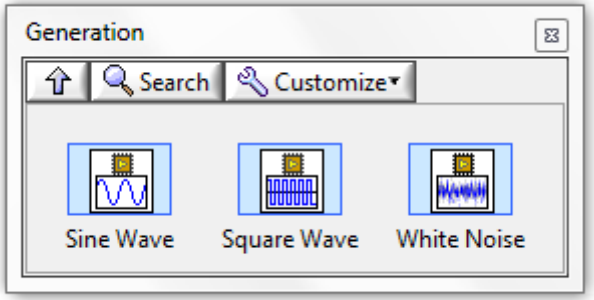


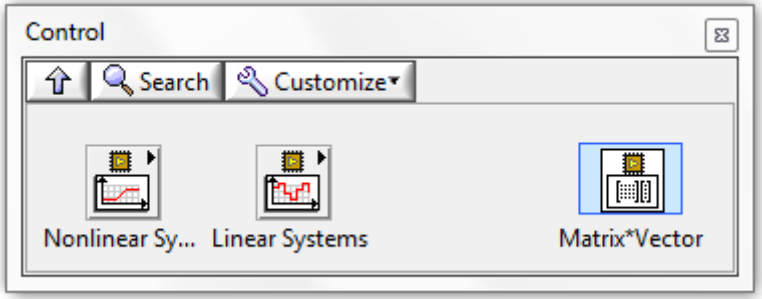
Standard Palette Items

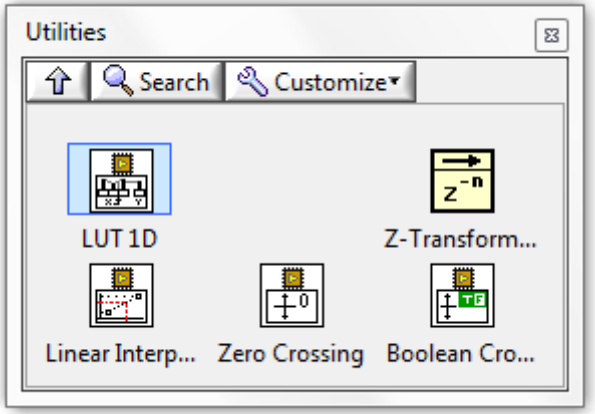


LabVIEW FPGA IP



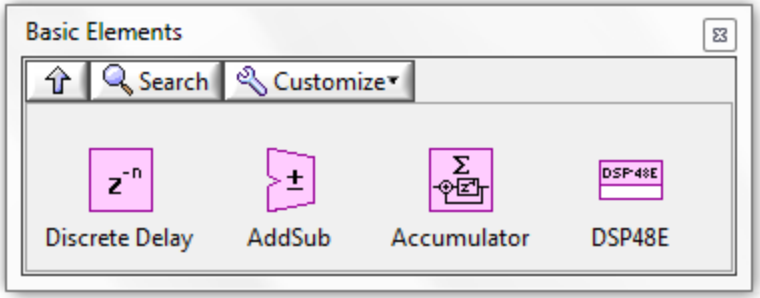


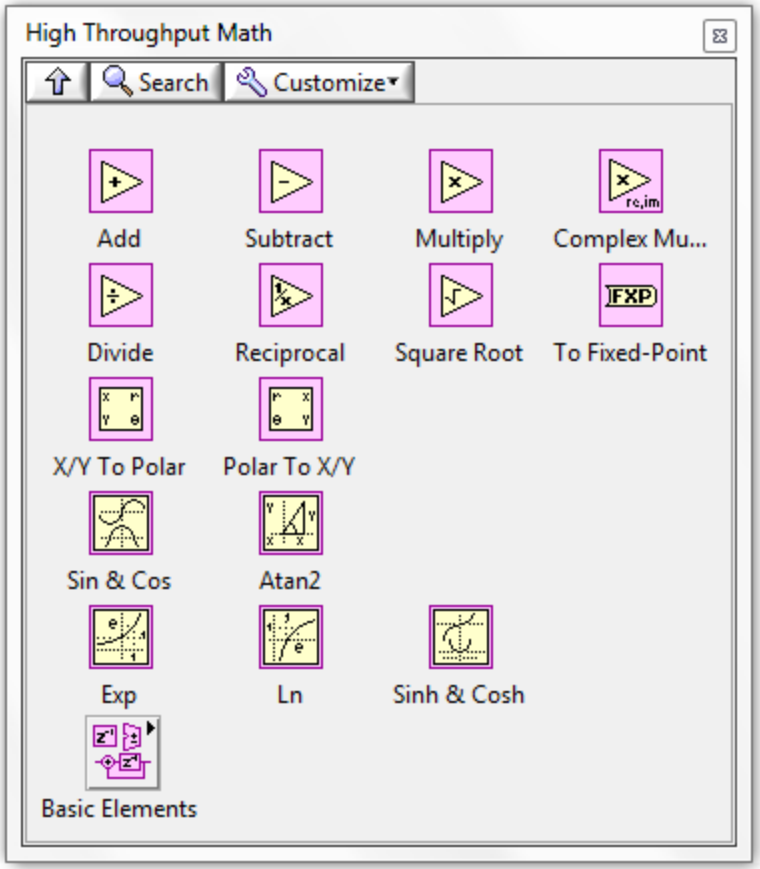




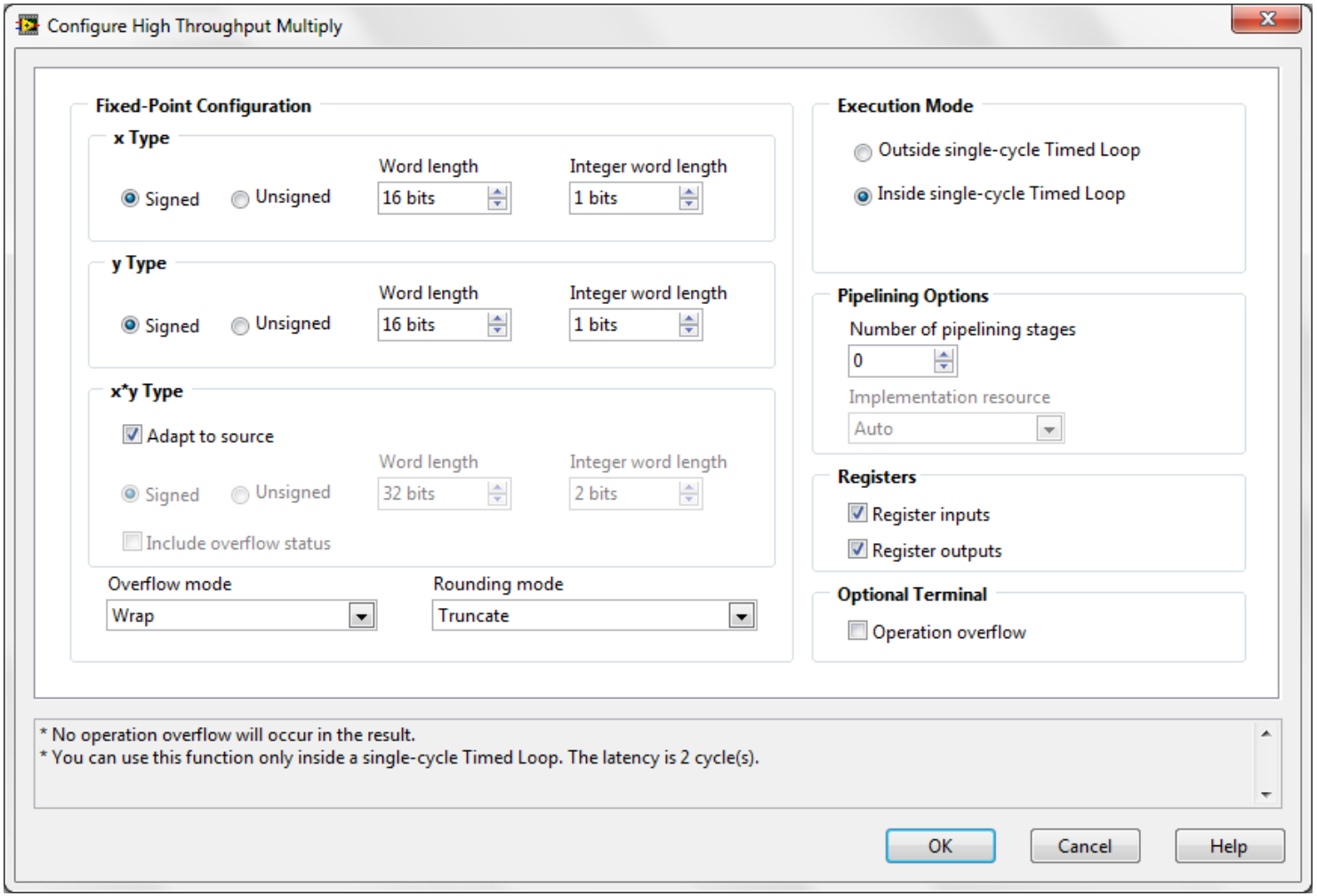
LabVIEW High Throughput Math

- Uses specific FPGA hardware resources when available
- Supports pipelining and handshaking



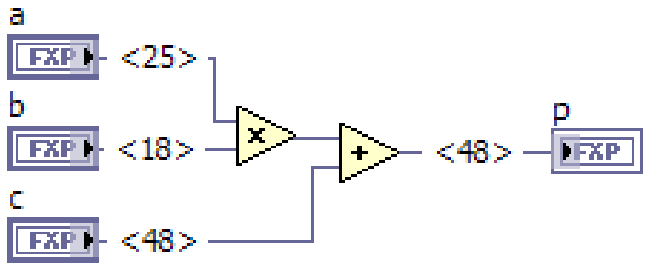


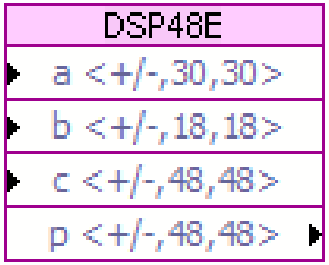
LabVIEW High Throughput Math

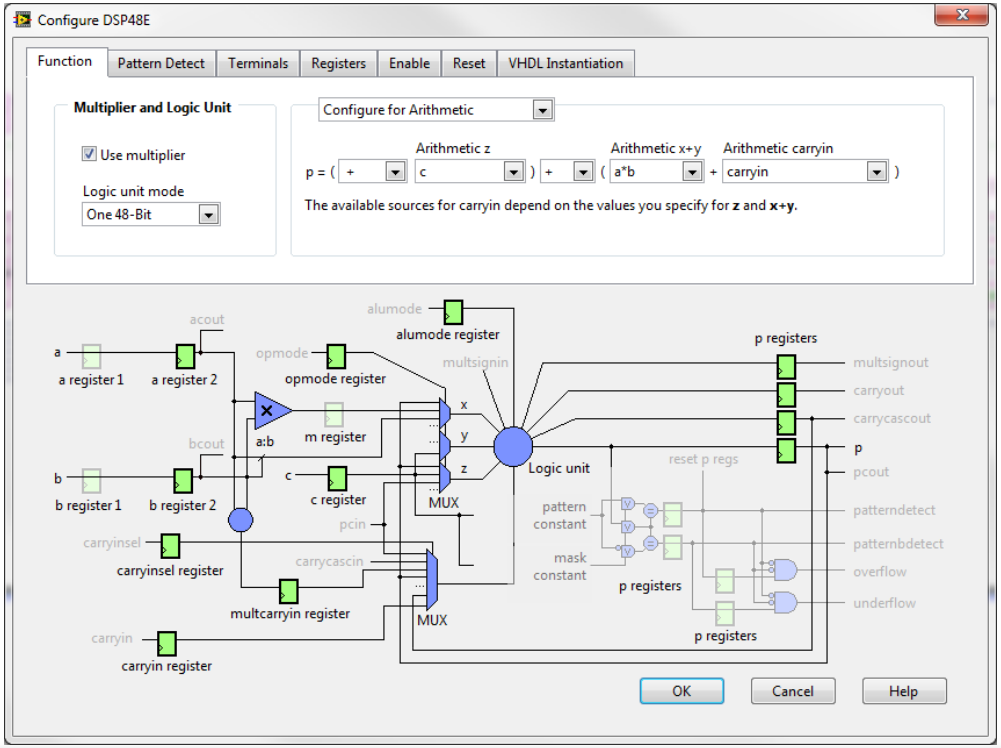


DSP48E Slices

- Directly configure FPGA DSP resources
- Optimize hardware for a specific application





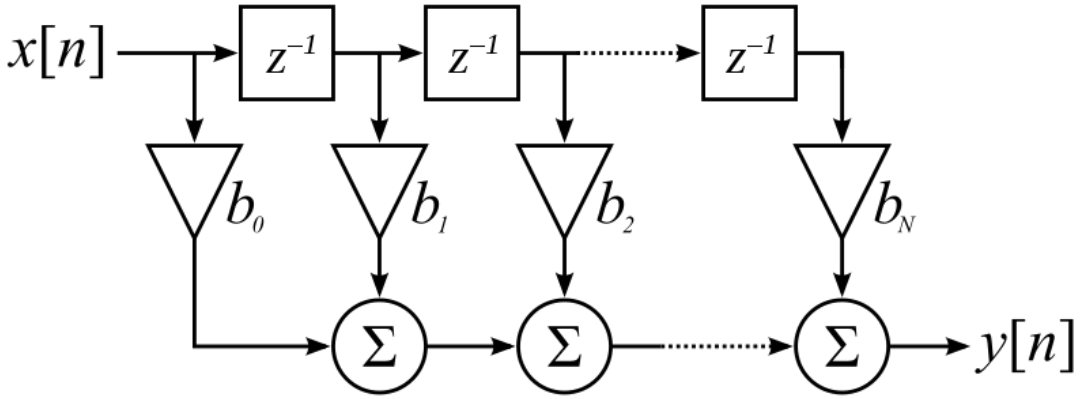


CASE STUDY: FIR FILTER

FIR Filter

Case Study

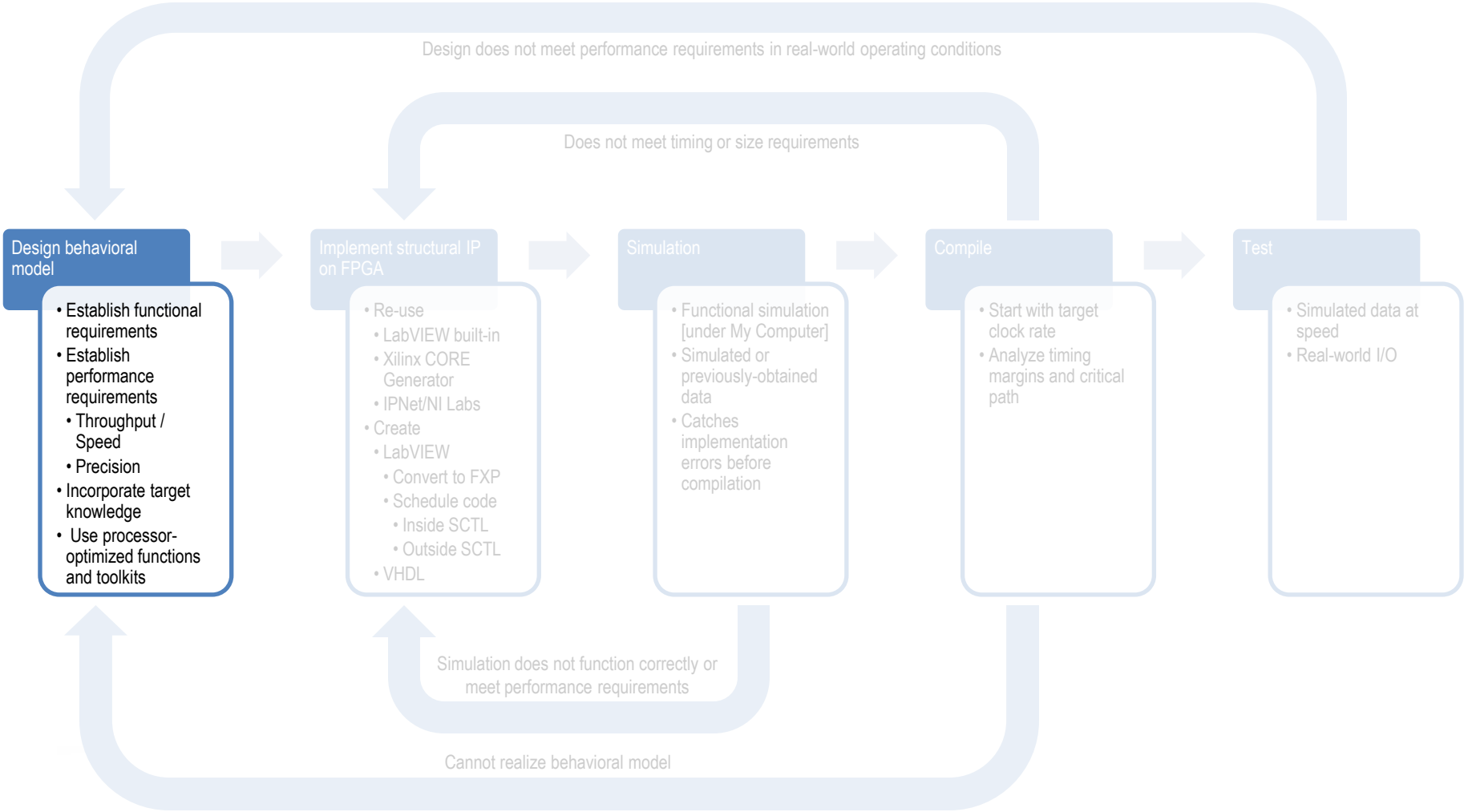
A **Finite Impulse Response (FIR)** filter is a type of a signal processing filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time.



$$y[n] = b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N]$$

Credit: Wikipedia

FPGA Design Flow



FIR Filter

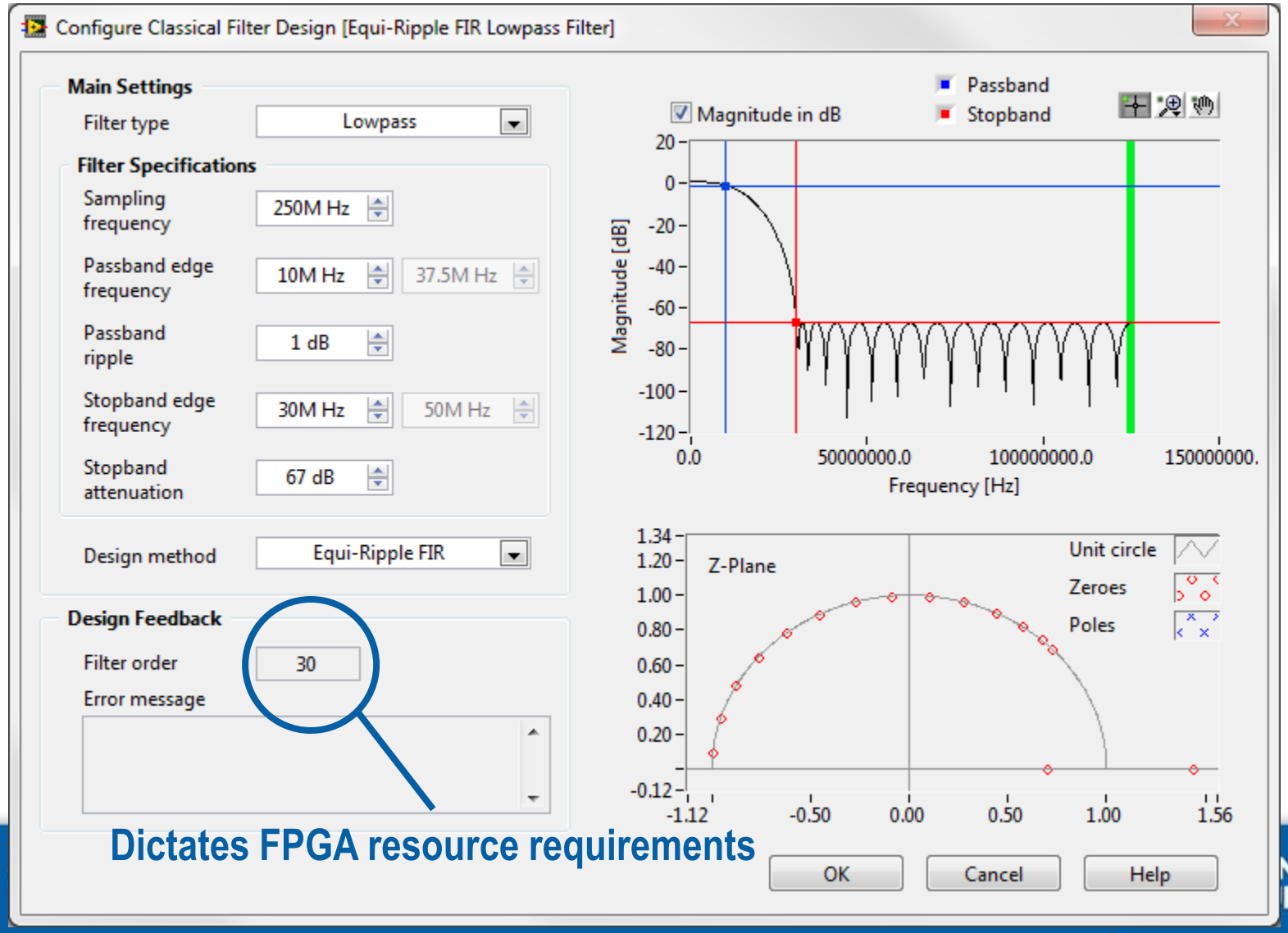
Requirements

Low-pass filter incoming signal on NI 5761 adapter module on FPGA

- Filter specifications
  - 10 MHz bandwidth
  - < 1 dB ripple
  - > 30 MHz stop band
  - > 65 dB attenuation
- 250 MS/s, 14-bit ADC

Generate Filter Coefficients

Digital Filter Design Toolkit

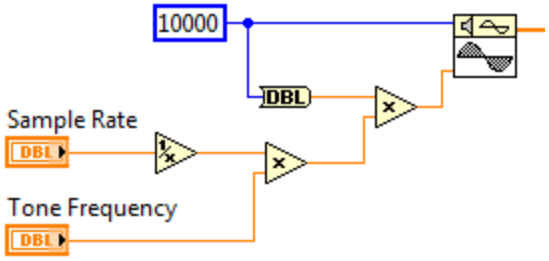


Dictates FPGA resource requirements

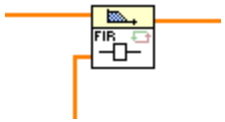
Behavioral Model

Demo

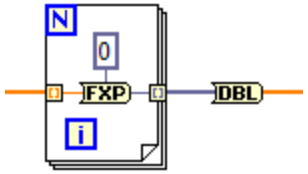
Simulate Test Signal



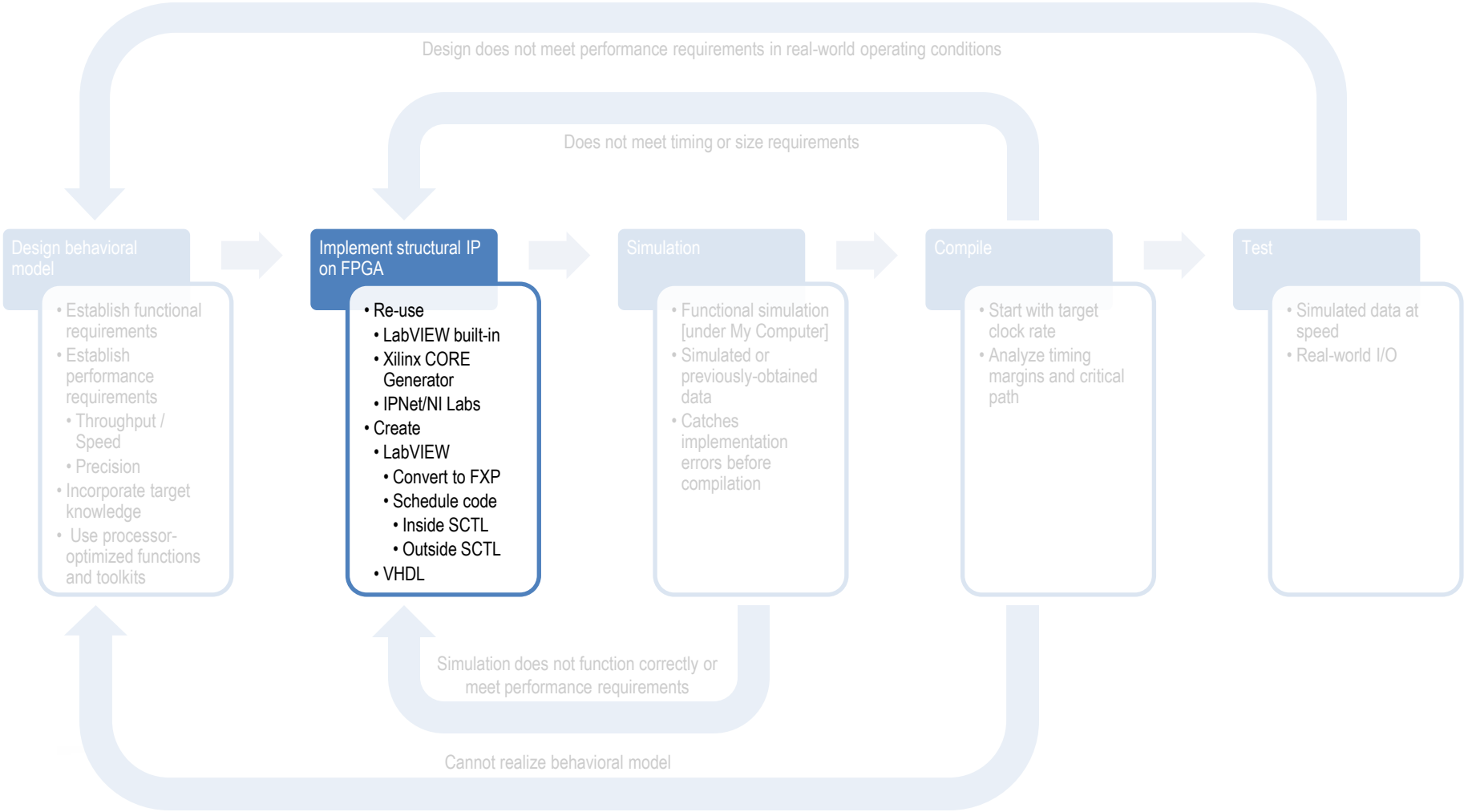
Filter Signal



Convert to Fixed-point

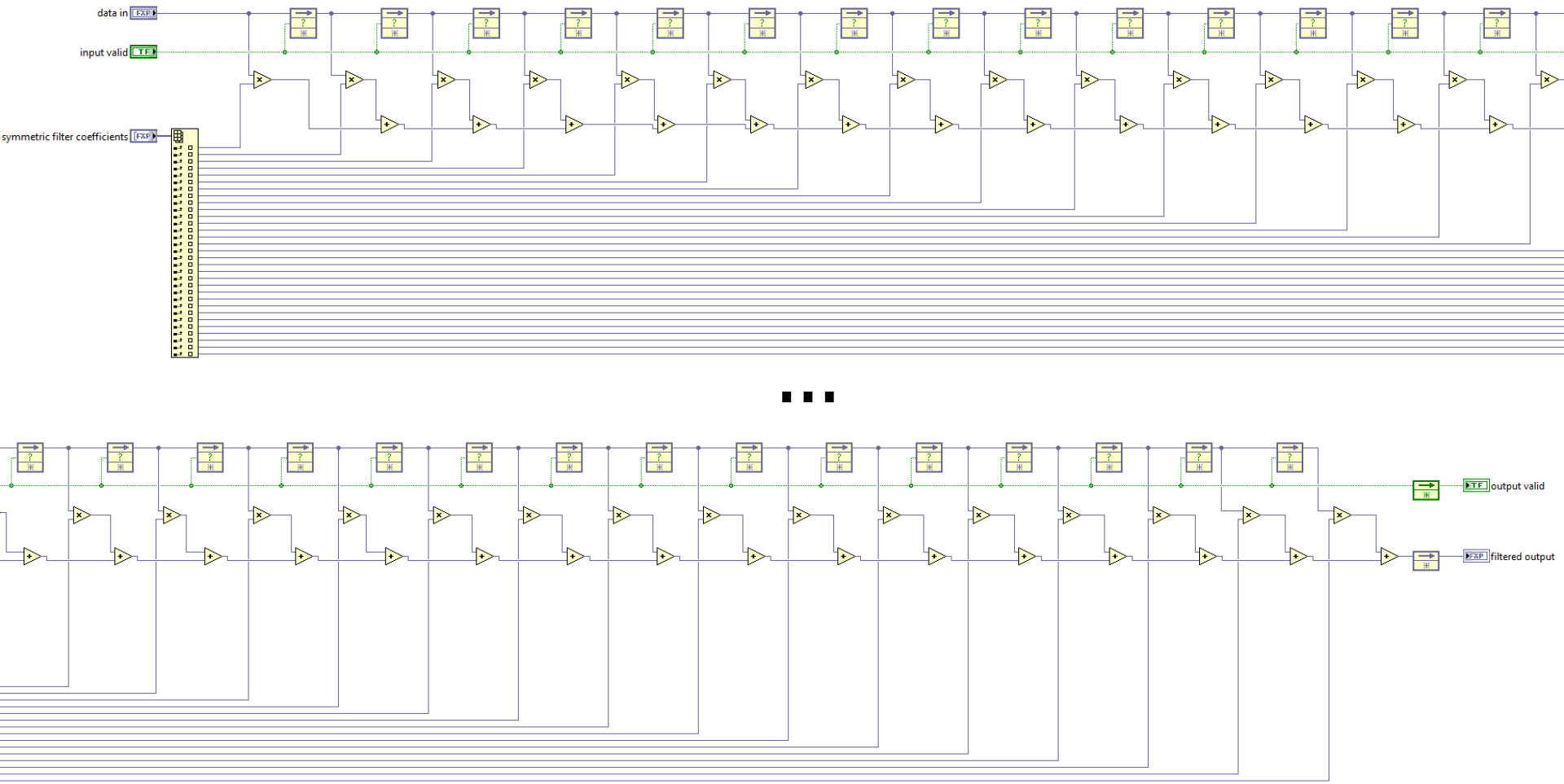


FPGA Design Flow

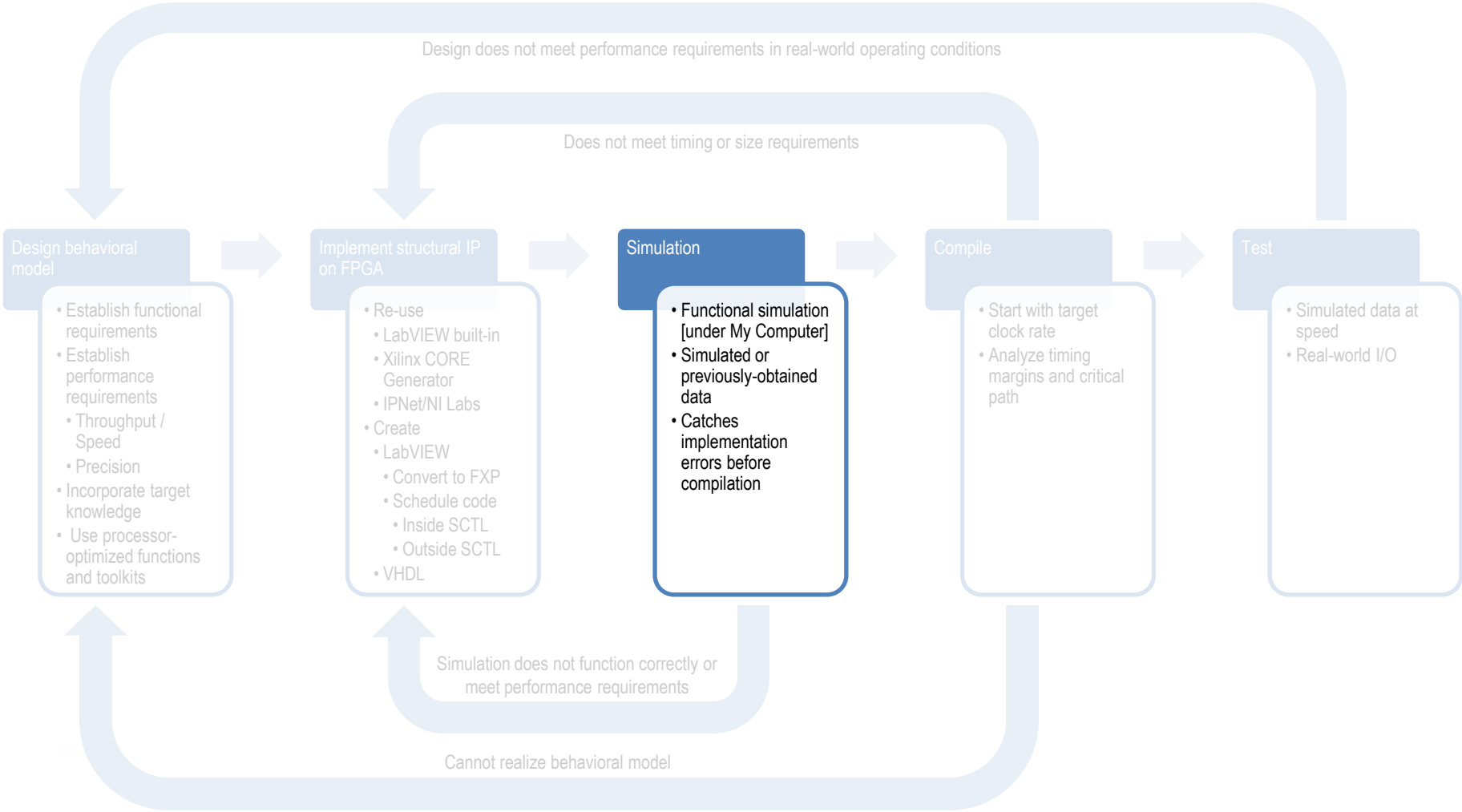


BASIC IMPLEMENTATION

Basic LabVIEW FPGA Implementation

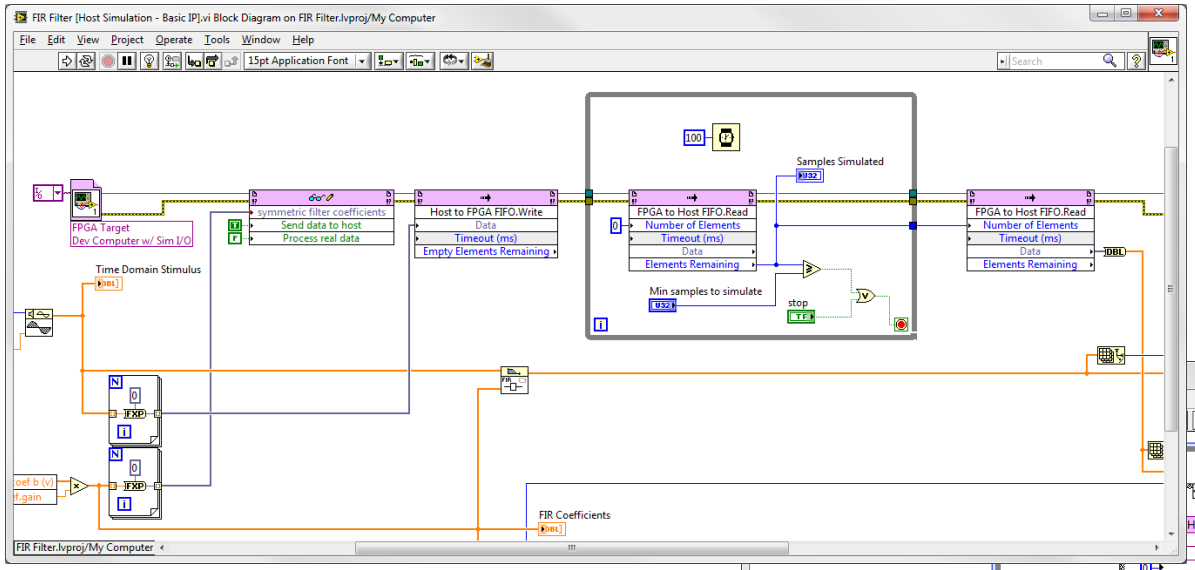


FPGA Design Flow



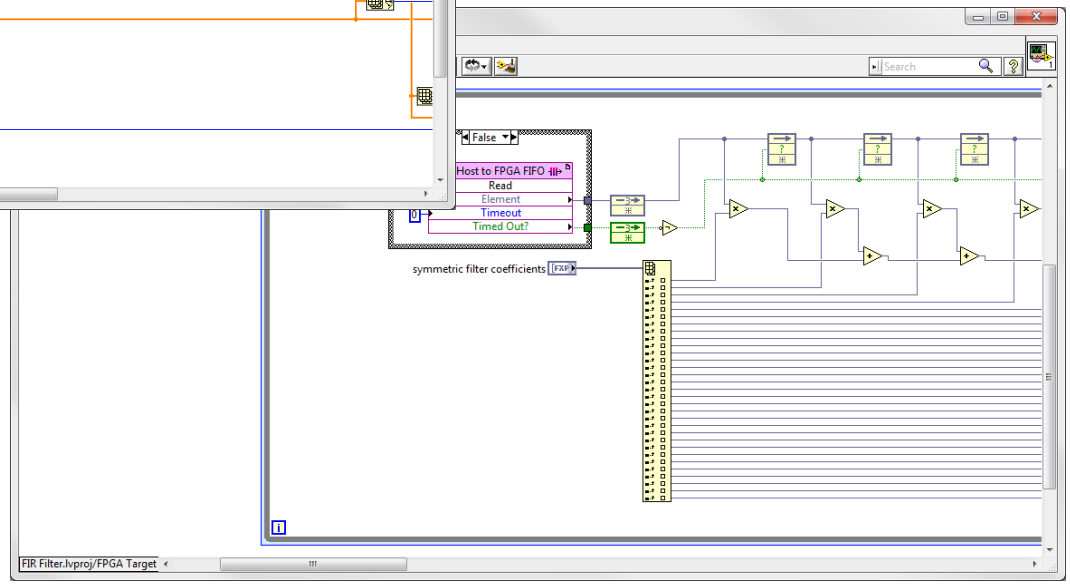
Simulation:

Dataflow-Accurate Functional Simulation Demo



Host Memory

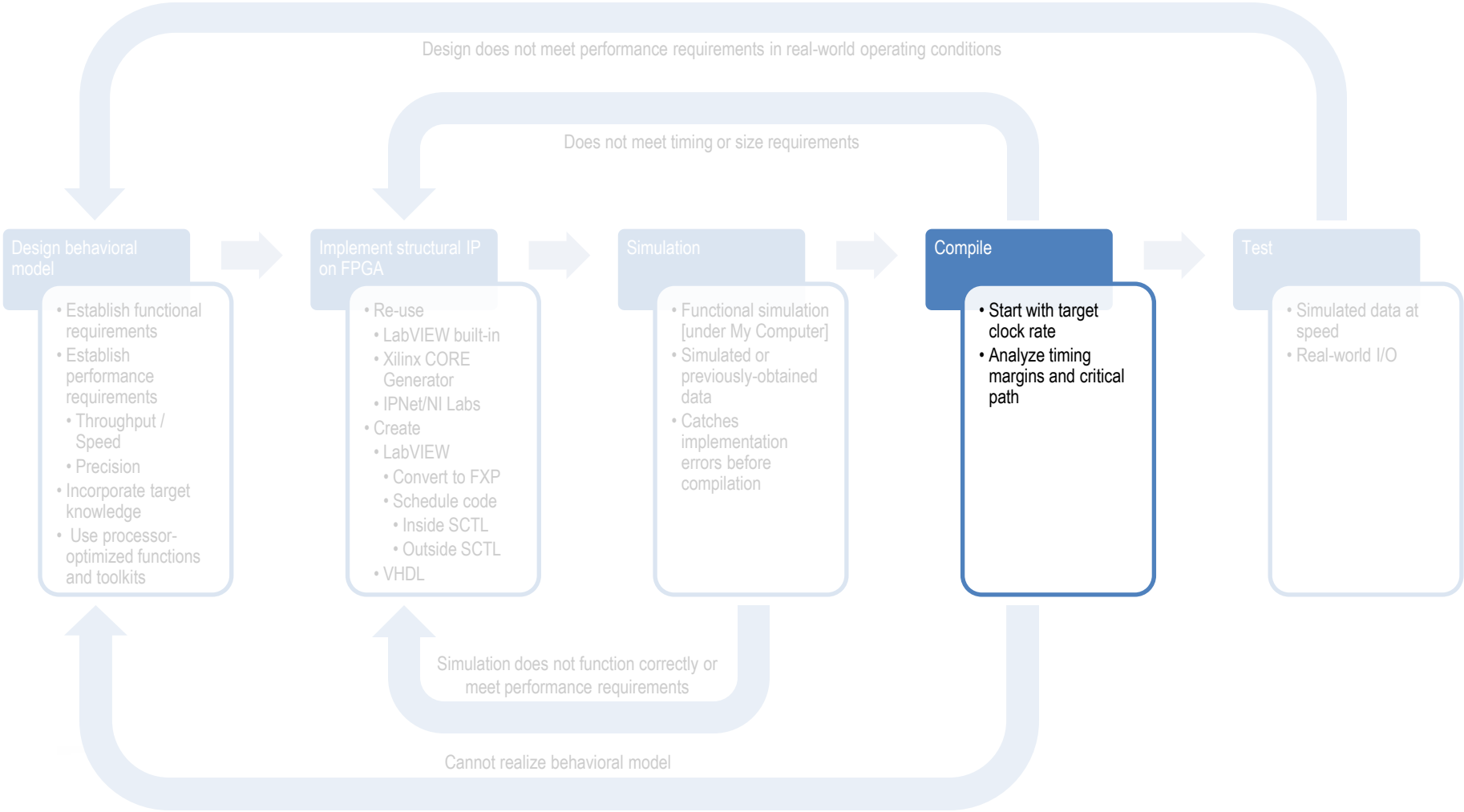
FPGA VI



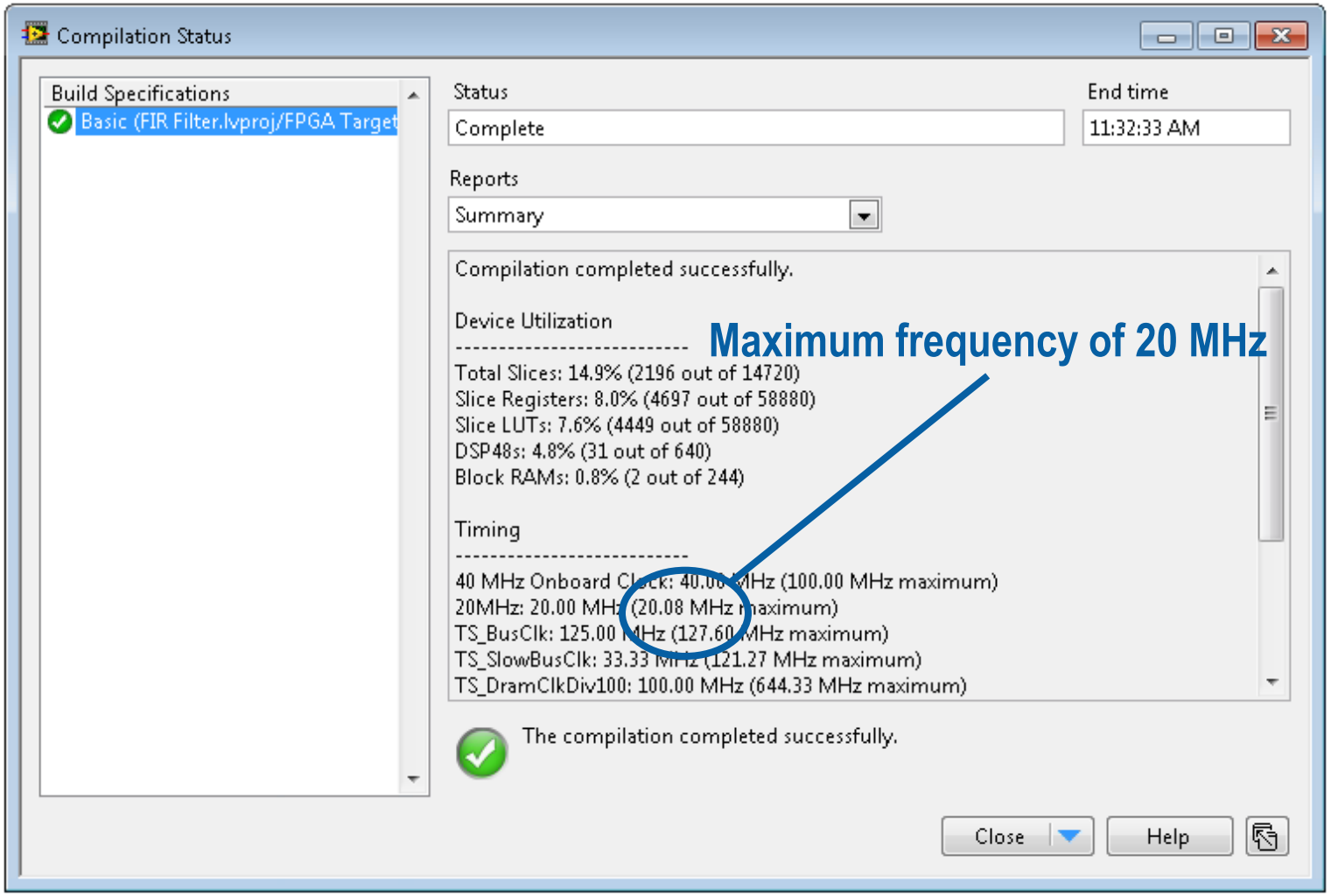
Host VI

Host Memory

FPGA Design Flow



Basic Compile Results



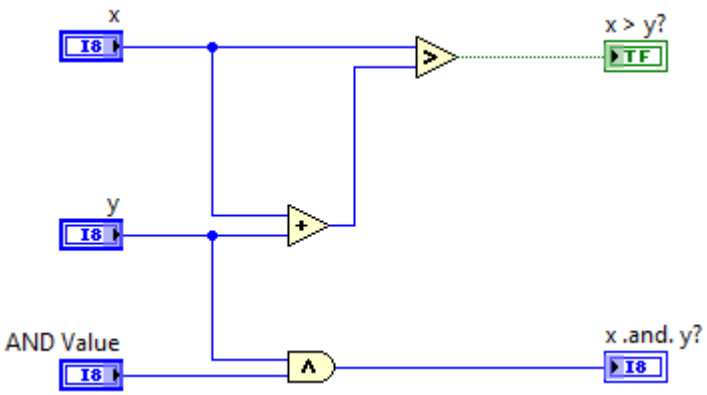
Design Resource Utilization

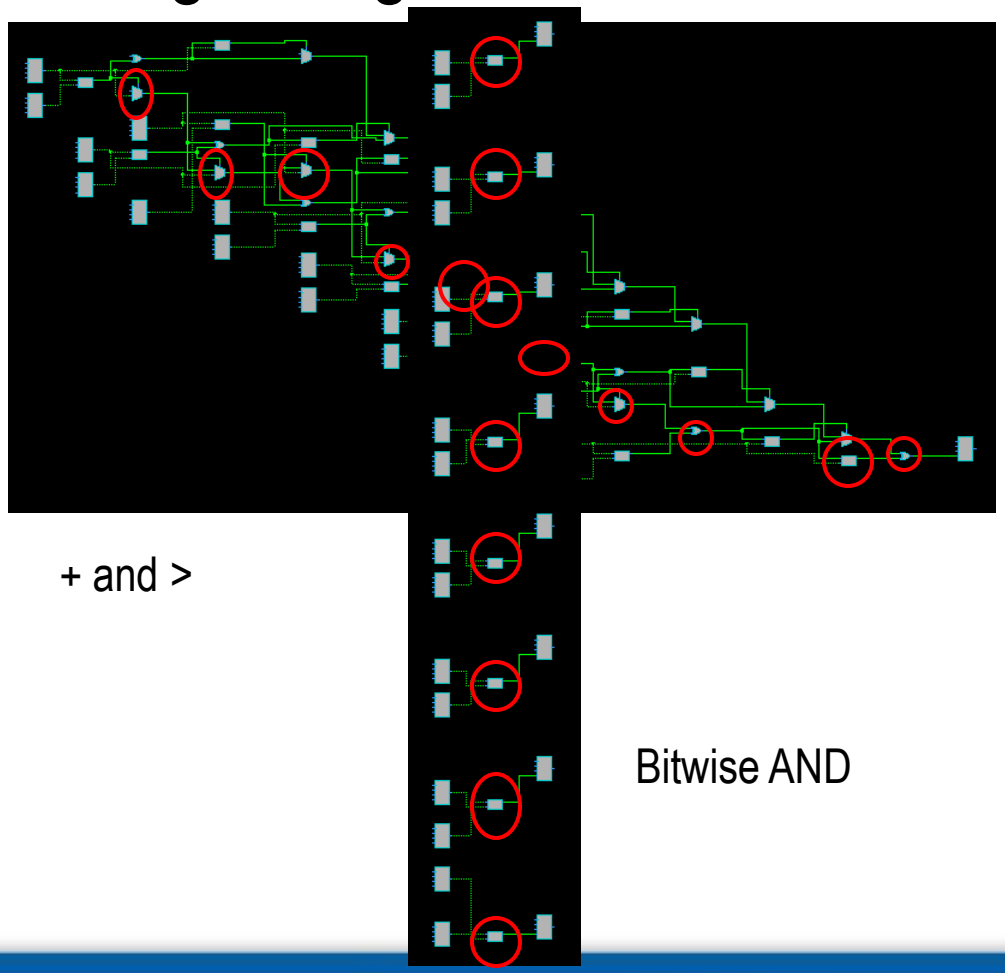
Implementation	Max. Freq.	Slices	Registers	LUTs	DSP	BRAMs
Basic	20	2196	4697	4449	31	0

Take-away: Not fast enough to filter 250 MHz ADC data

Logic Delay

- The number of consecutive logic stages varies based on logic implementation

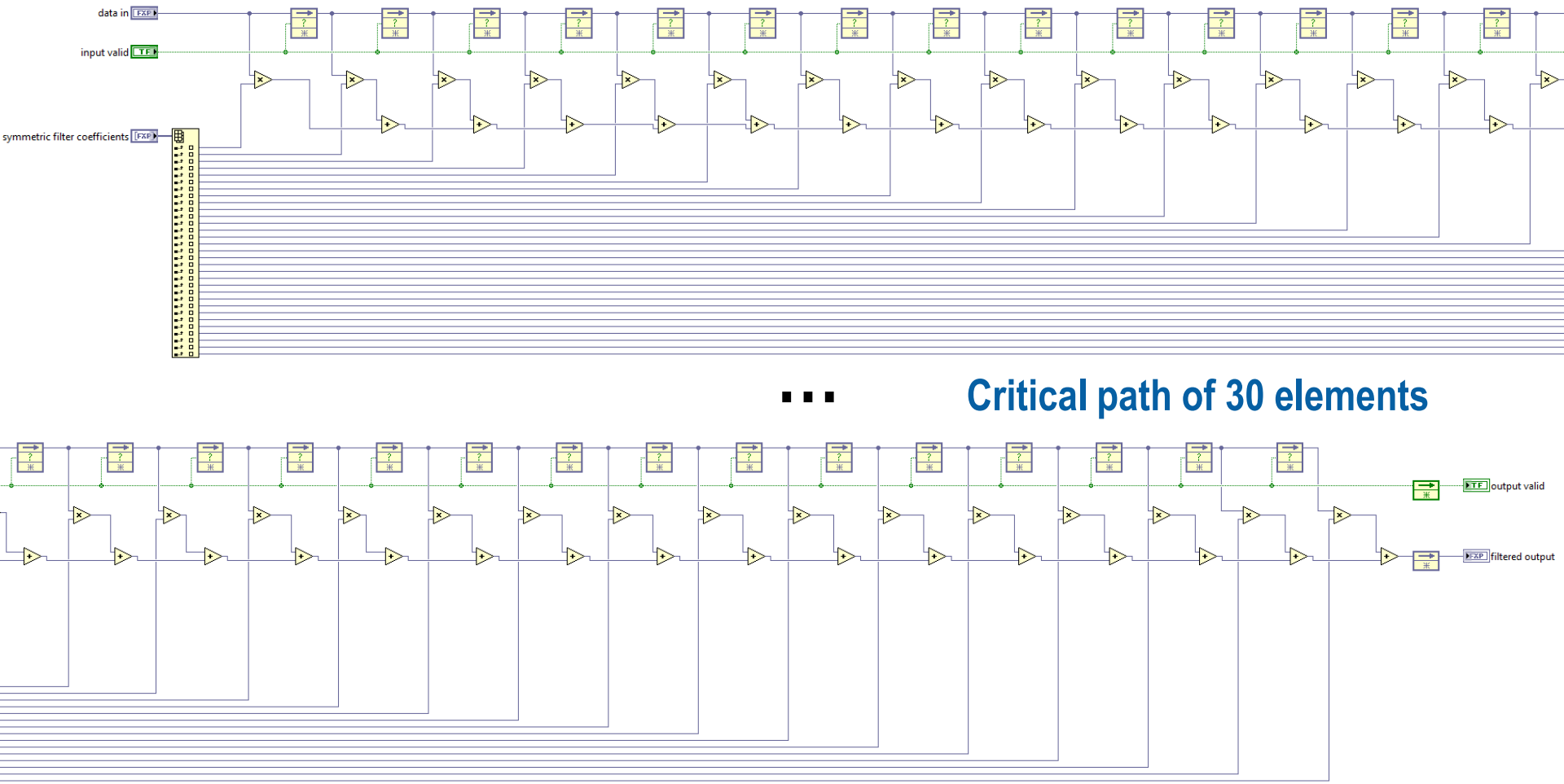




+ and >

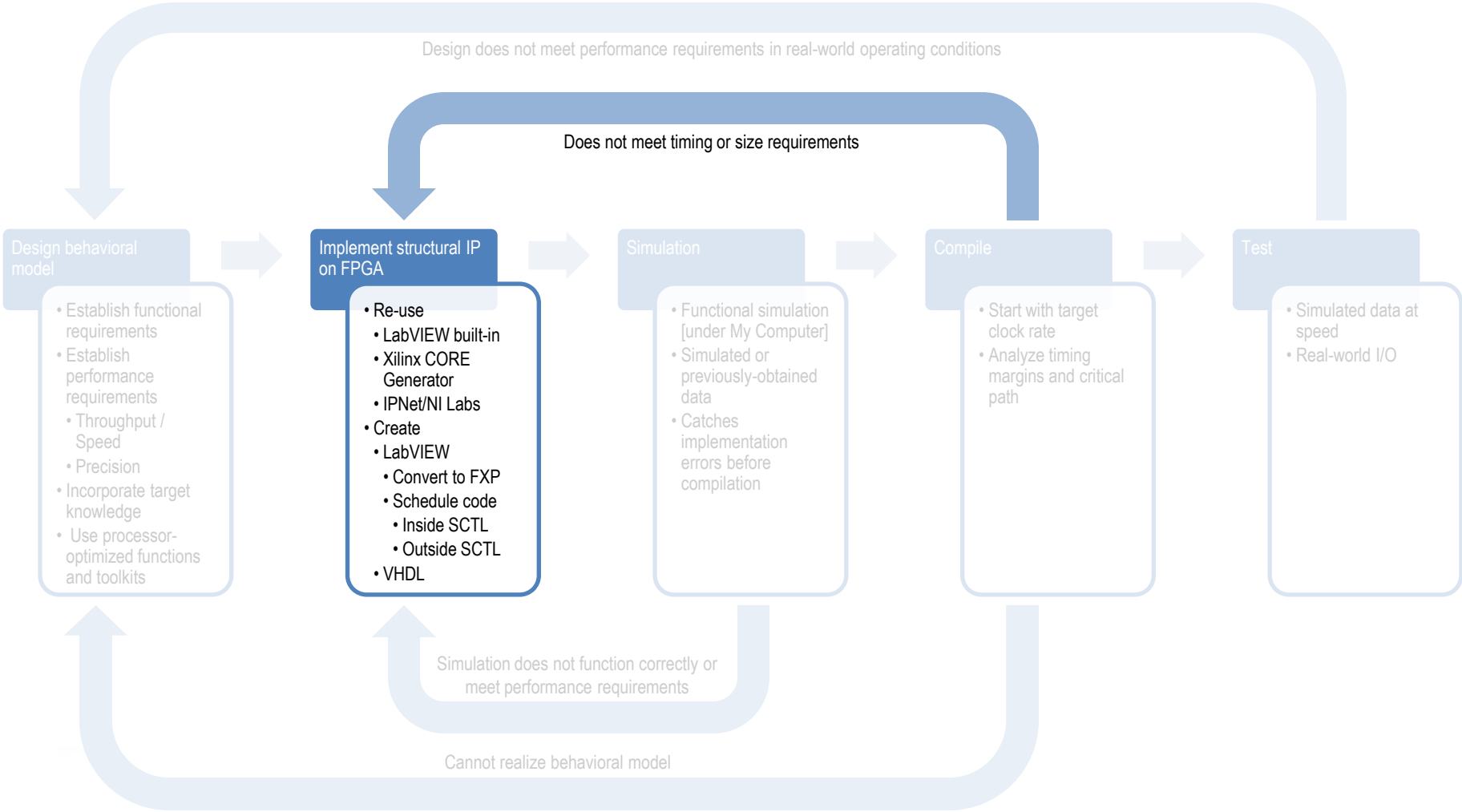
Bitwise AND

Basic LabVIEW FPGA Implementation

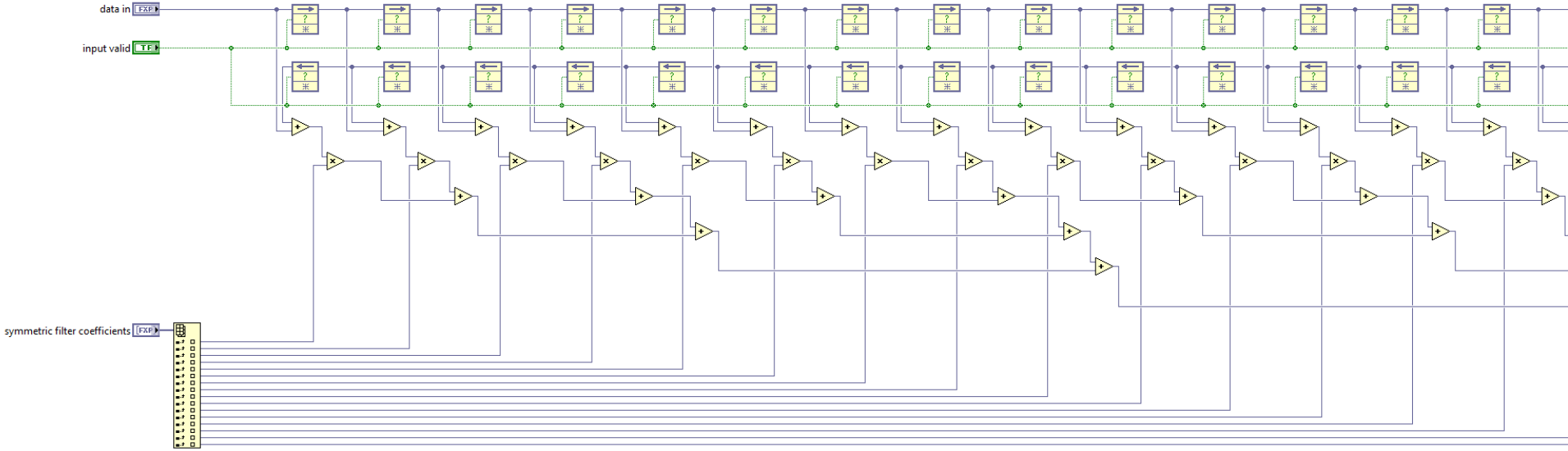


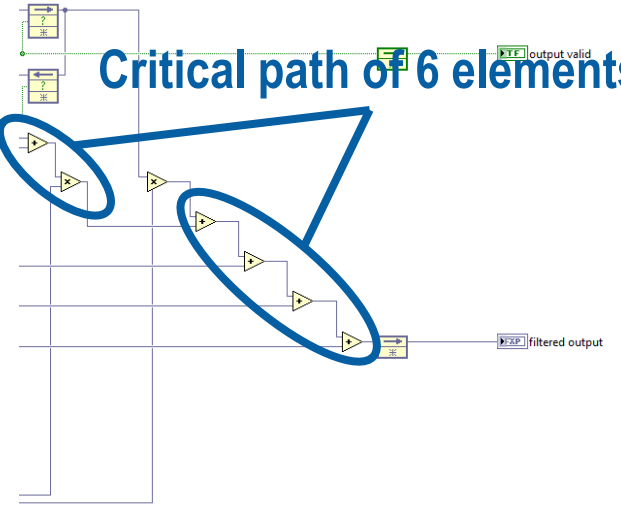
▪ ▪ ▪ Critical path of 30 elements

FPGA Design Flow



Symmetric Optimization





Critical path of 6 elements

- Designed to reduce critical path length
- Effort to reduce resources – half the number of multipliers
- Fewer coefficients needed

Design Resource Utilization

Implementation	Max. Freq.	Slices	Registers	LUTs	DSP	BRAMs
Basic	20	2196	4697	4449	31	0
Adder Tree	70	2209	4697	4404	31	0
Symmetric Optimization	70	2034	4193	3888	16	0

Take-away: Fewer coefficients and higher rate, but not fast enough for this application

METHODS TO INCREASE THROUGHPUT

Methods to Increase Throughput

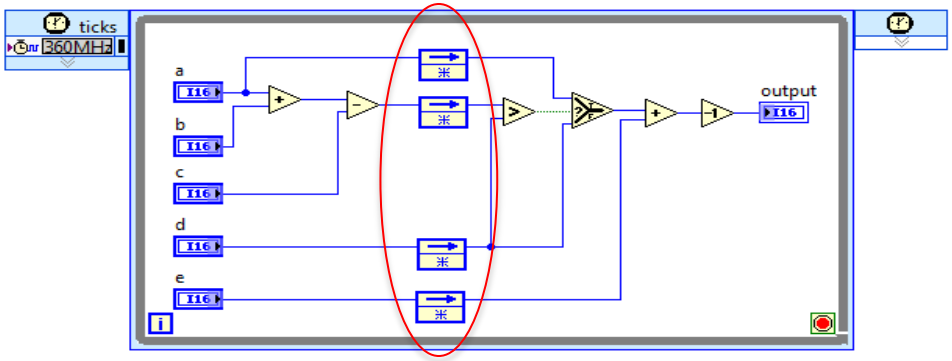
Parallelize algorithm

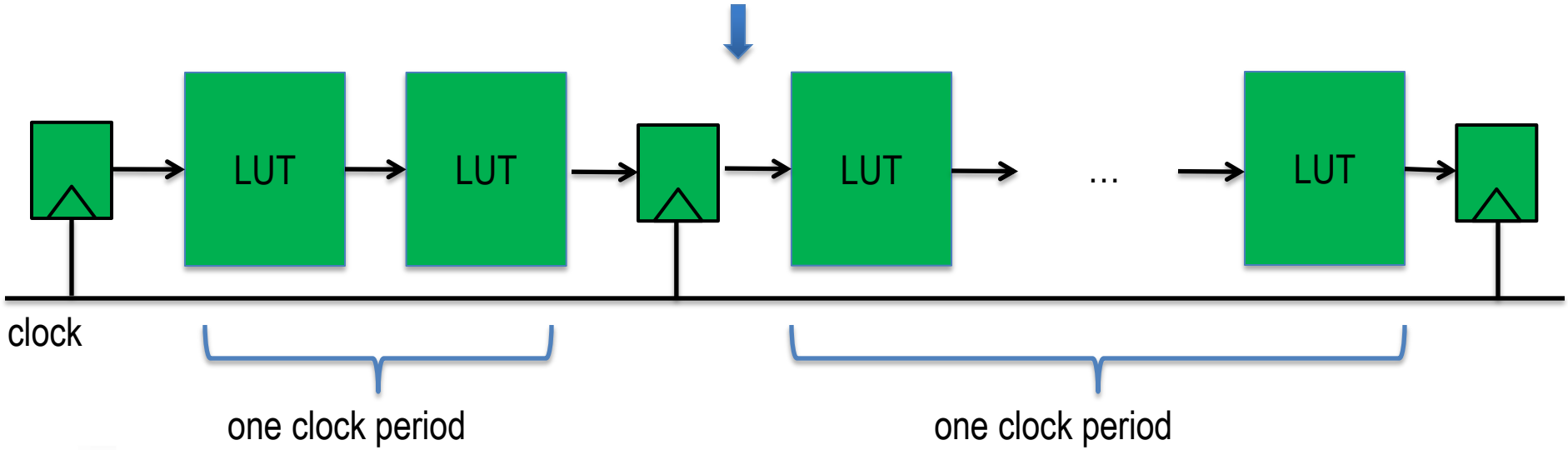
- Easier if no consecutive data dependencies
- Challenges
  - Not all algorithms are easy to port
  - Increases resource utilization
- Extremely algorithm dependent

Increase Clock Rate

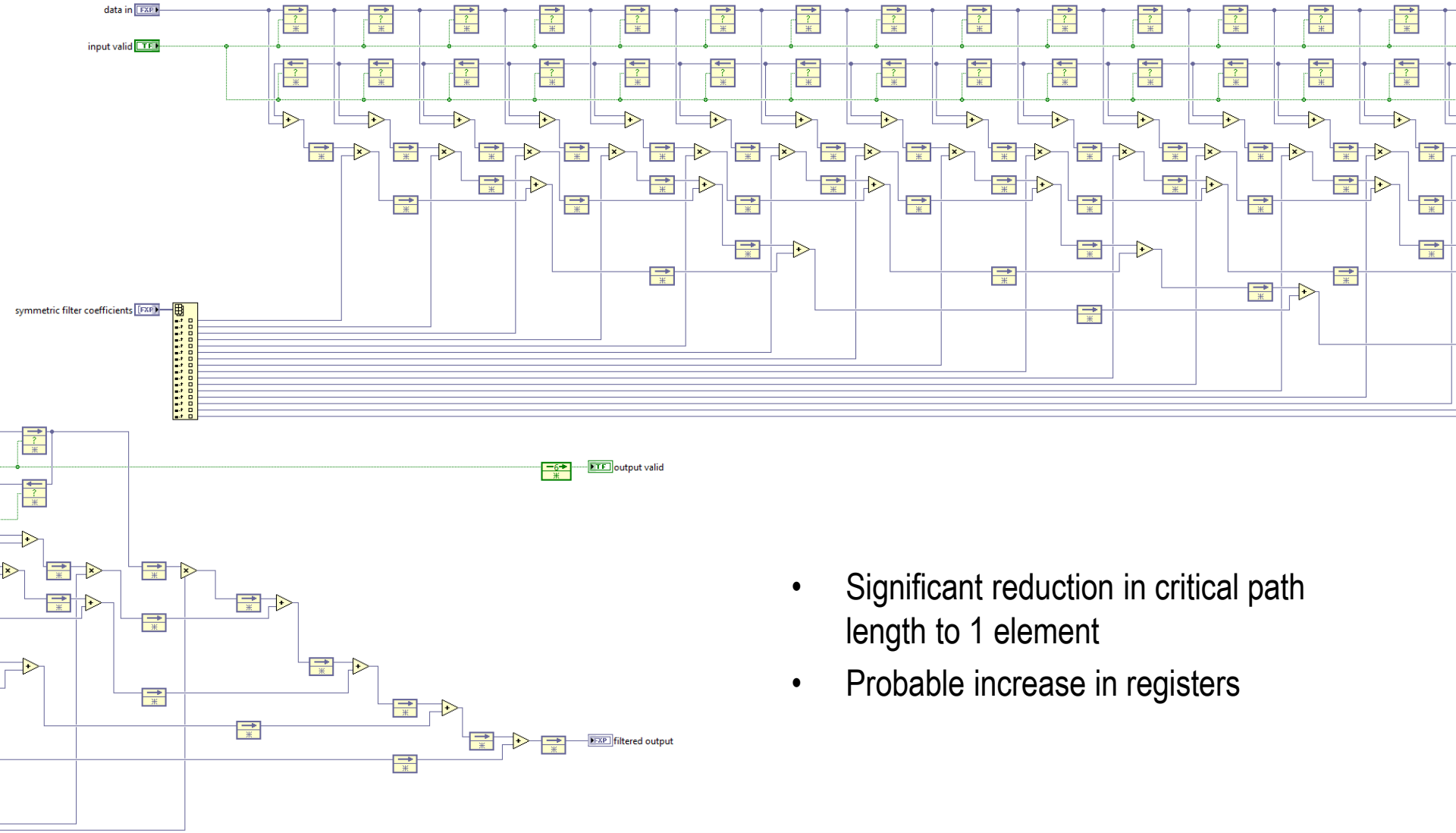
- Easier if there is timing margin in design
- Challenges
  - Dependency on critical path
  - May require pipelining – possible resource and latency increase

Pipelining


ticks
360MHz
a
b
c
d
e
output
I16
i


LUT
LUT
LUT
...
LUT
clock
one clock period
one clock period

Pipelined



- Significant reduction in critical path length to 1 element
- Probable increase in registers

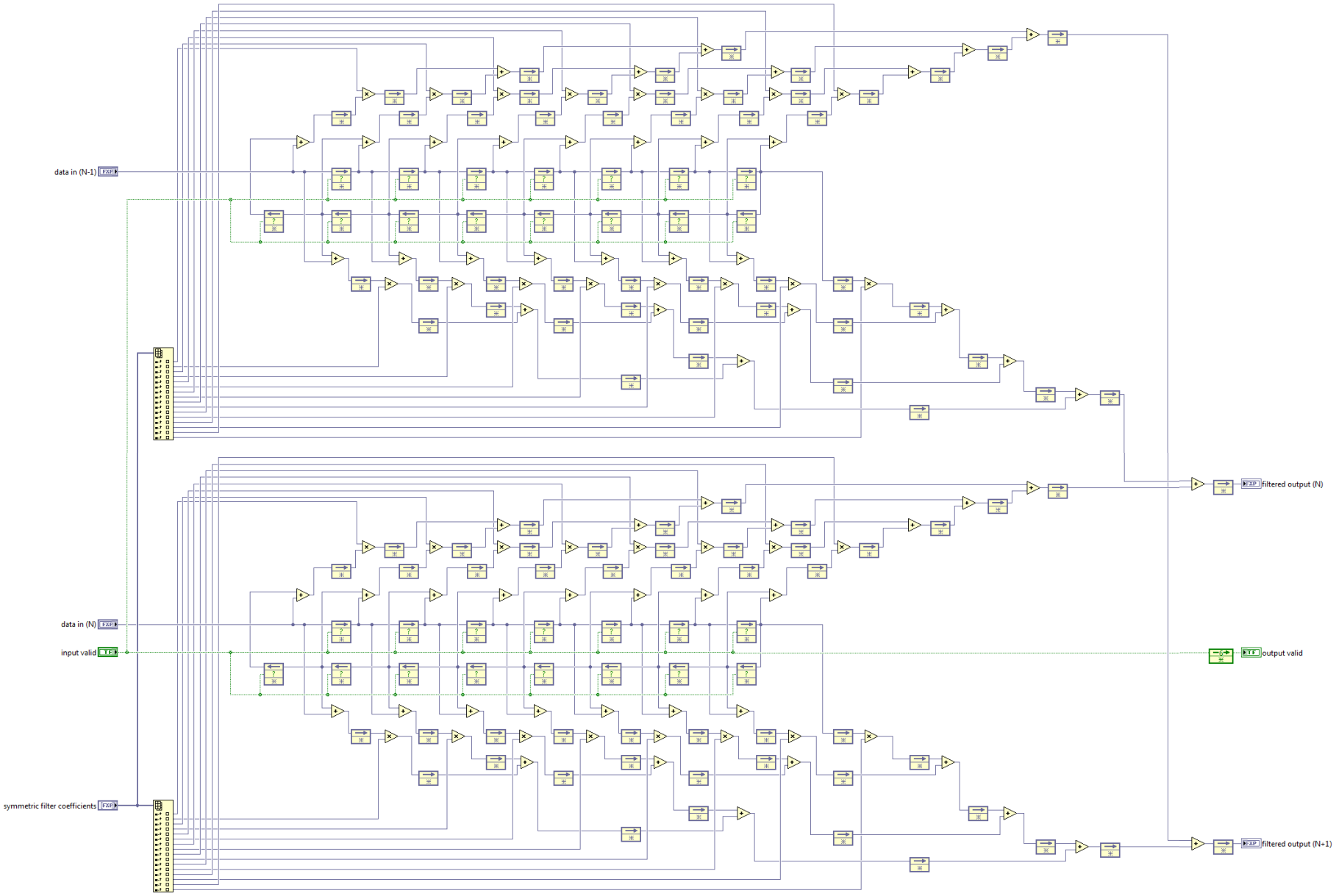
Design Resource Utilization

Implementation	Max. Freq.	Slices	Registers	LUTs	DSP	BRAMs
Basic	20	2196	4697	4449	31	0
Adder Tree	70	2209	4697	4404	31	0
Symmetric Optimization	70	2034	4193	3888	16	0
Pipelined	160	2208	5478	3890	16	0

Take-away: Much faster than before, but not fast enough, and many more registers

Parallel

- Throughput double clock rate
- Significant increase in resources

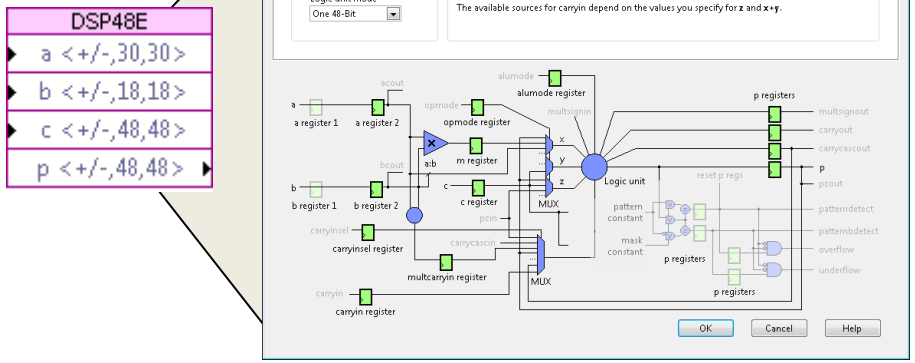


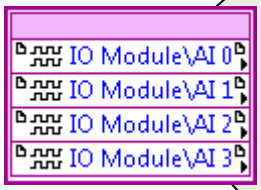
Design Resource Utilization

Implementation	Max. Freq.	Slices	Registers	LUTs	DSP	BRAMs
Basic	20	2196	4697	4449	31	0
Adder Tree	70	2209	4697	4404	31	0
Symmetric Optimization	70	2034	4193	3888	16	0
Pipelined	160	2208	5478	3890	16	0
Parallel	160	2622	6849	4746	32	0

Take-away: Significant increase in resources, but meets throughput needs! Implications on downstream logic.

Performance – Hardware Specific Items





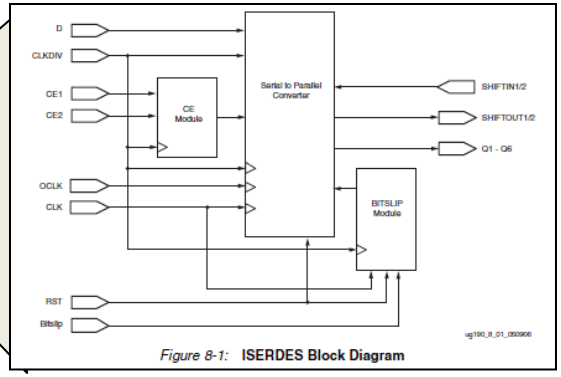
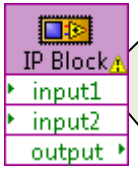
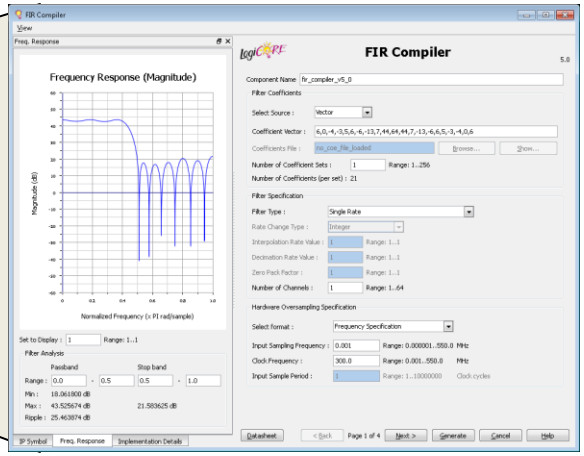


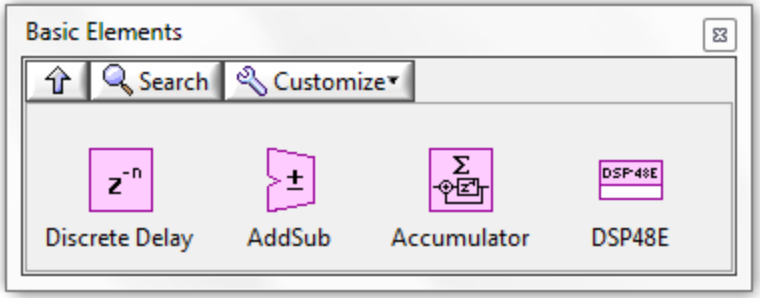
Figure 8-1: ISERDES Block Diagram

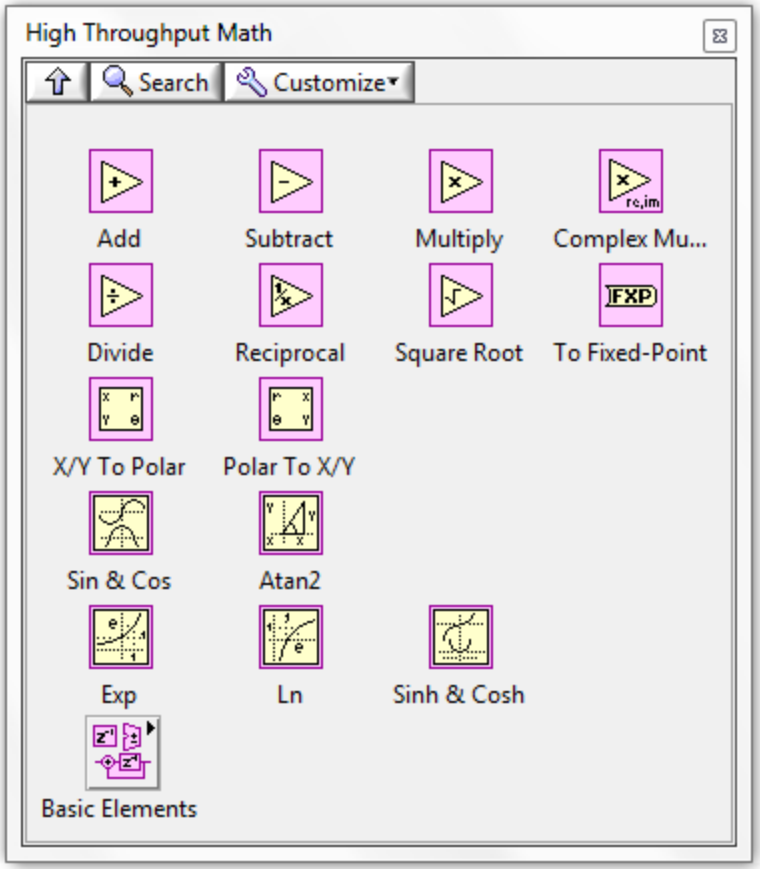




LabVIEW High Throughput Math

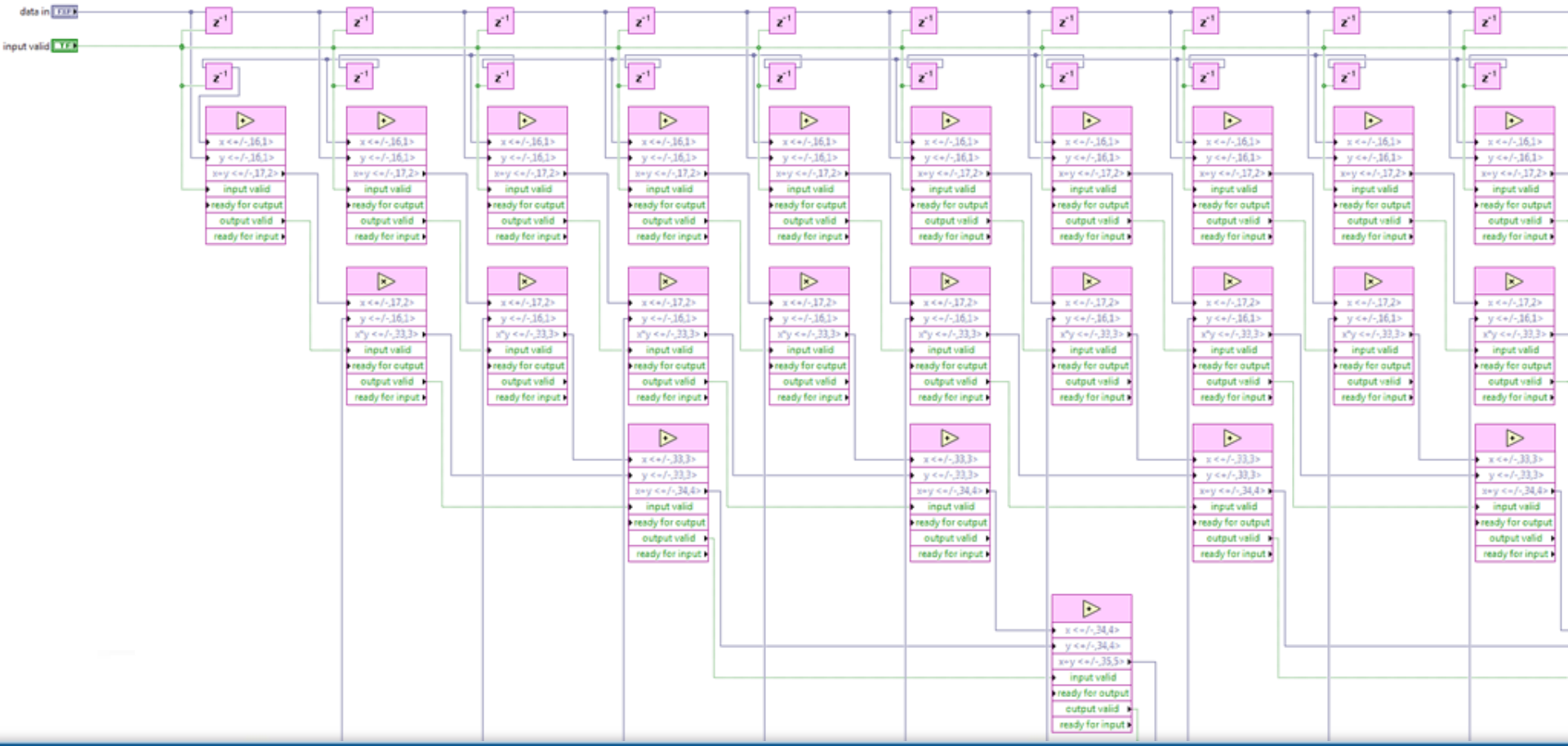
- Uses specific FPGA hardware resources when available
- Supports pipelining and handshaking





High Throughput Math

- Return to serial operation
- Effort to increase clock rate
- Registers and handshaking in nodes



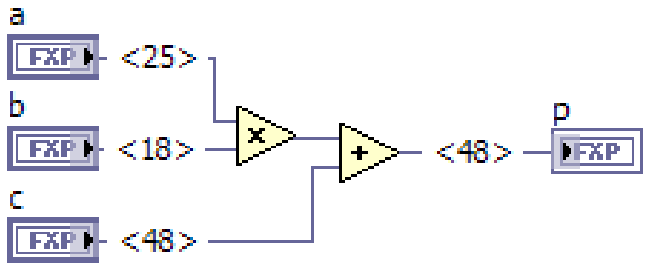
Design Resource Utilization

Implementation	Max. Freq.	Slices	Registers	LUTs	DSP	BRAMs
Basic	20	2196	4697	4449	31	0
Adder Tree	70	2209	4697	4404	31	0
Symmetric Optimization	70	2034	4193	3888	16	0
Pipelined	160	2208	5478	3890	16	0
Parallel	160	2622	6849	4746	32	0
High Throughput Math	240	2401	6003	4417	16	0

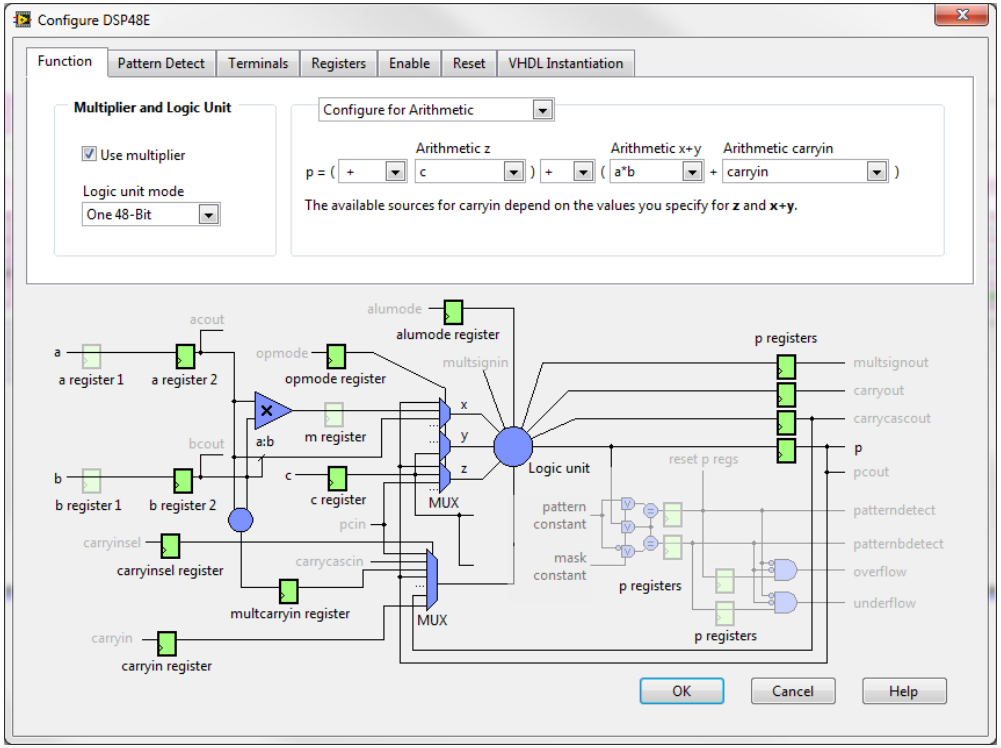
Take-away: Fewer DSP slices and faster clock rate, but not fast enough for serial (non-parallel) operation

DSP48E Slices

- Directly configure FPGA DSP resources
- Optimize hardware for a specific application

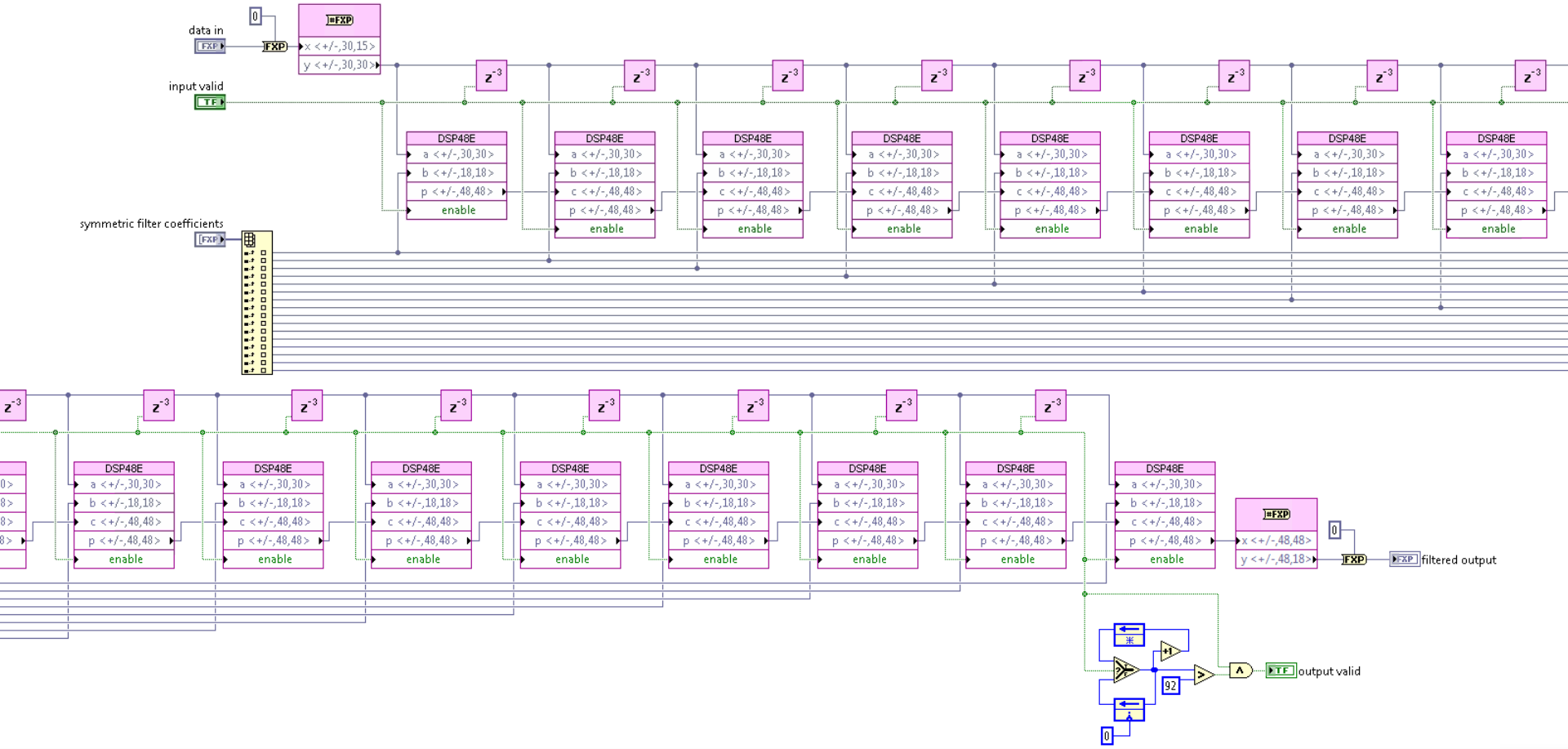


DSP48E
a <+/-,30,30>
b <+/-,18,18>
c <+/-,48,48>
p <+/-,48,48>



DSP 48E Node

- Take advantage of optimized, dedicated hardware
- Uses internal pipelining capabilities



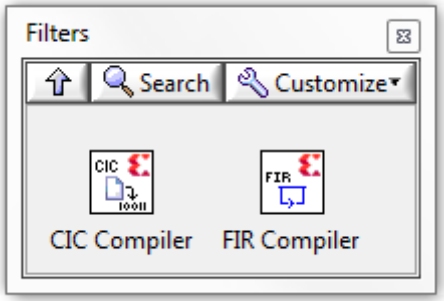
Design Resource Utilization

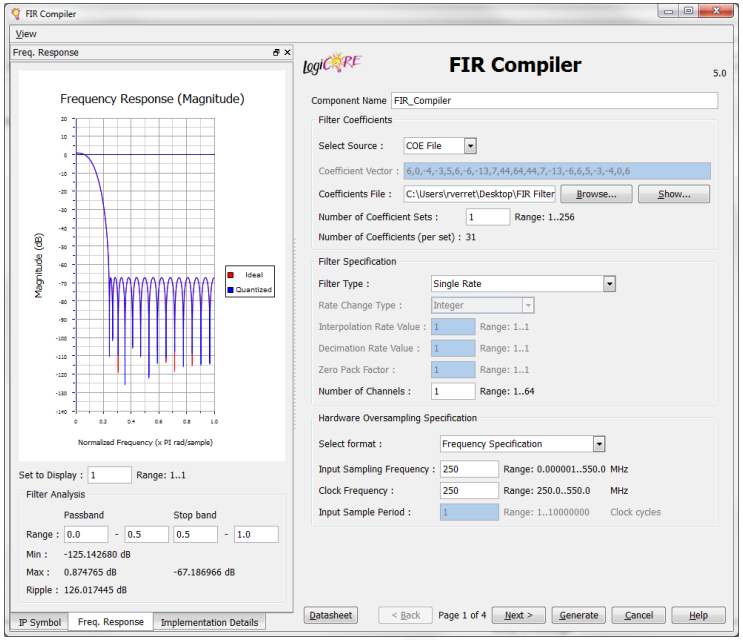
Implementation	Max. Freq.	Slices	Registers	LUTs	DSP	BRAMs
Basic	20	2196	4697	4449	31	0
Adder Tree	70	2209	4697	4404	31	0
Symmetric Optimization	70	2034	4193	3888	16	0
Pipelined	160	2208	5478	3890	16	0
Parallel	160	2622	6849	4746	32	0
High Throughput Math	240	2401	6003	4417	16	0
DSP 48E Node	260	2304	4666	5000	31	0

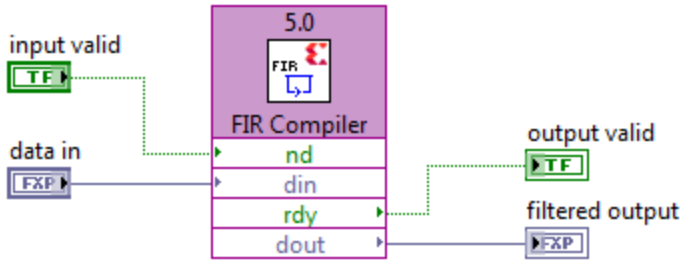
Take-away: Increase in DSP resources, but meets serial rate requirement!

XILINX IP

Xilinx CORE Generator IP







- Extension of IP Integration Node
- Automatically launches CORE Generator IP customization
  - Entire design in a single node
  - Extremely optimized
- Available in **LabVIEW FPGA 2011**

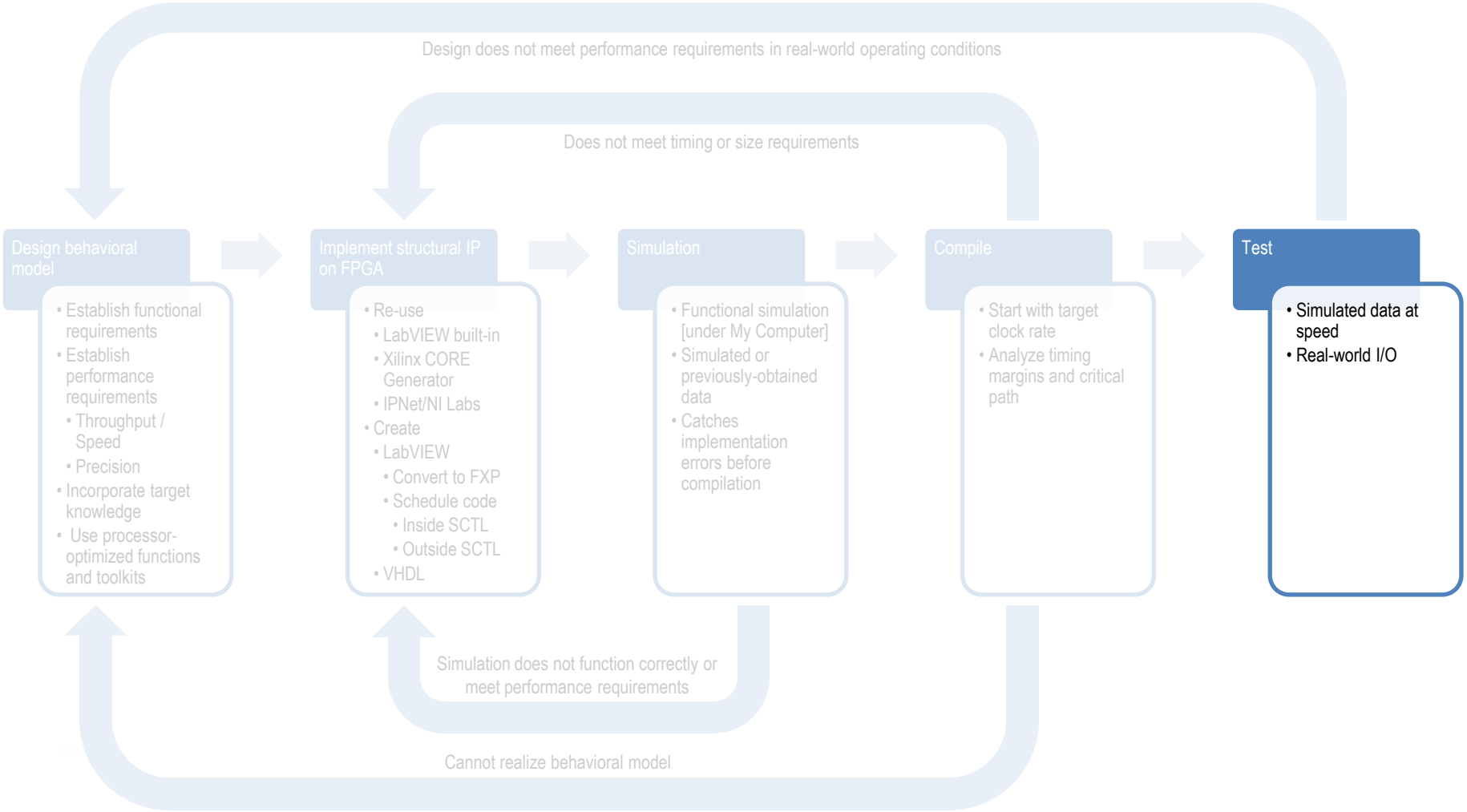
Design Resource Utilization

Implementation	Max. Freq.	Slices	Registers	LUTs	DSP	BRAMs
Basic	20	2196	4697	4449	31	0
Adder Tree	70	2209	4697	4404	31	0
Symmetric Optimization	70	2034	4193	3888	16	0
Pipelined	160	2208	5478	3890	16	0
Parallel	160	2622	6849	4746	32	0
High Throughput Math	240	2401	6003	4417	16	0
DSP 48E Node	260	2304	4666	5000	31	0
Xilinx CORE Generator	300	1827	4061	3710	16	0

Take-away: Most efficient implementation and highest clock rate, with minimal user programming

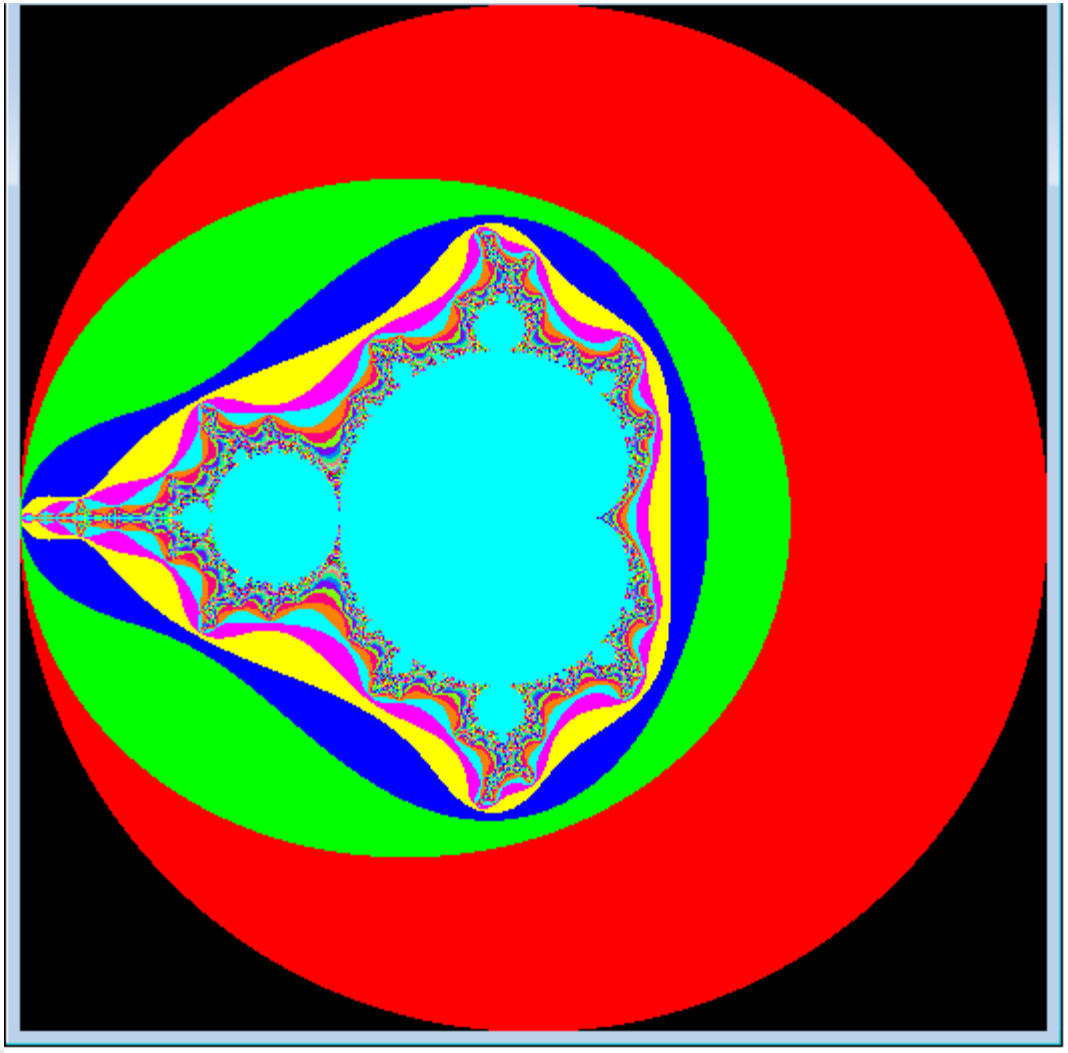
FPGA Design Flow

Test Demo



Demo Time

What is That?



Mandelbrot Set – Back to the Happy 80's

- C and Z being Complex
- Z(0) = 0
- Z(*n*) = Z(*n-1*)^2 + C
- Until |Z(*n*)| < 2
- 'Display' *n* as function of C
- ... So given a complex value C, how many iterations *n* it takes before |Z(*n*)| < 2.
- *n* may be huge or even infinite so limited to a max.

Summary

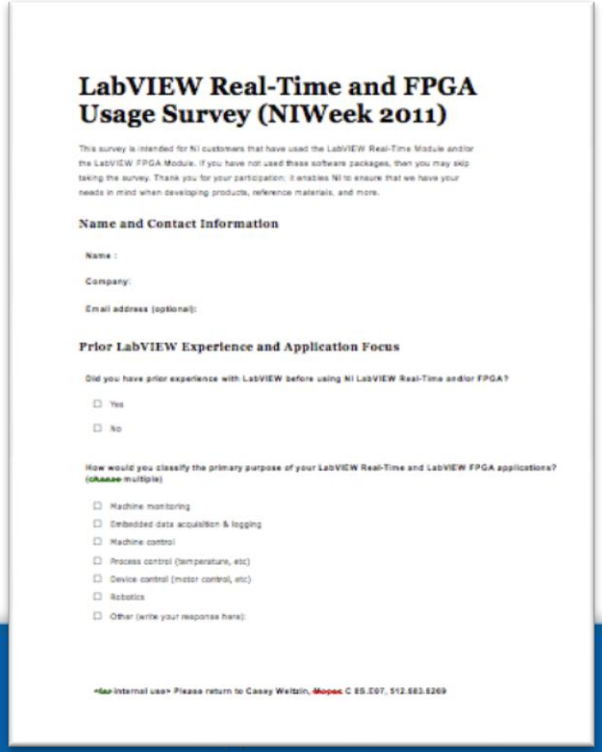
- Suggested algorithm design flow helps minimize time spent waiting on compiles and makes debugging easier
- Use existing IP if possible:
  - LabVIEW code
  - ni.com/ipnet
  - ni.com/labs
  - Xilinx CORE Generator
  - VHDL
- When building IP, consider optimization options:
  - Algorithm-specific optimizations
  - Reduce combinatorial path lengths (tree vs. serial)
  - Pipelining
  - Parallelize algorithm
  - Leverage algorithm-specific FPGA hardware (DSP nodes, I/O resources, etc.)

We need your feedback.

- If you have used LabVIEW Real-Time or FPGA, please fill out the 5-minute paper survey

- This will help NI better define our products and documentation according to your applications and experience

- You can skip taking the survey if you've already taken it, or if you have never used LabVIEW Real-Time or FPGA



LabVIEW Real-Time and FPGA Usage Survey (NIWeek 2011)

This survey is intended for NI customers that have used the LabVIEW Real-Time Module and/or the LabVIEW FPGA Module. If you have not used these software packages, then you may skip taking the survey. Thank you for your participation; it enables NI to ensure that we have your needs in mind when developing products, reference materials, and more.

Name and Contact Information

Name :

Company:

Email address (optional):

Prior LabVIEW Experience and Application Focus

Did you have prior experience with LabVIEW before using NI LabVIEW Real-Time and/or FPGA?

- ☐ Yes
- ☐ No

How would you classify the primary purpose of your LabVIEW Real-Time and LabVIEW FPGA applications? (~~Choose~~ multiple)

- ☐ Machine monitoring
- ☐ Embedded data acquisition & logging
- ☐ Machine control
- ☐ Process control (temperature, etc)
- ☐ Device control (motor control, etc)
- ☐ Robotics
- ☐ Other (write your response here):

~~For~~ Internal use> Please return to Casey Weltzin, ~~Mopac~~ C 85.207, 512.683.6269