

# FPGA Construction: The Art behind it

Arun Veeramani  
National Instruments

# Outline

- Digital design fundamentals
- What is FPGA?
- Steps for constructing FPGA
- Testing the design
- VHDL basics

# Digital Design Fundamentals

- Building blocks for ALL LOGIC ??



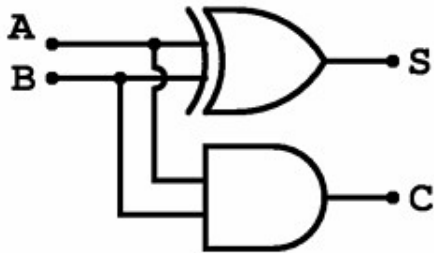
NAND Gate

Input A	Input B	Output
0	0	1
0	1	1
1	0	1
1	1	0

# Single Bit Adder

- **Half Adder**

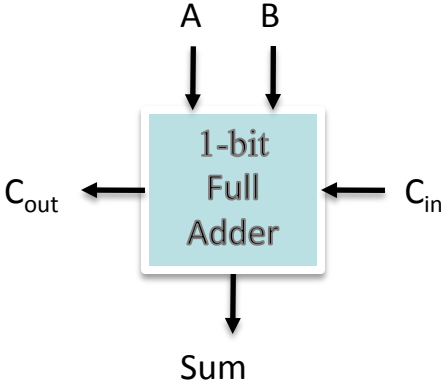
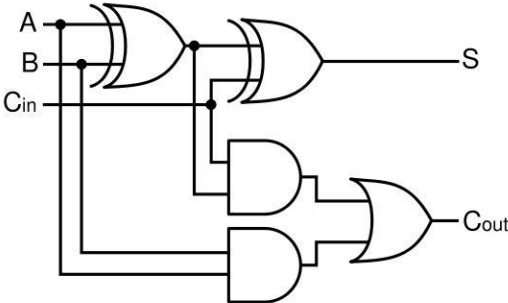
- $\text{Sum} = (\text{not } A).B + A.(\text{not } B)$
- $\text{Carry} = A.B$



Input A	Input B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

# Single Bit Adder

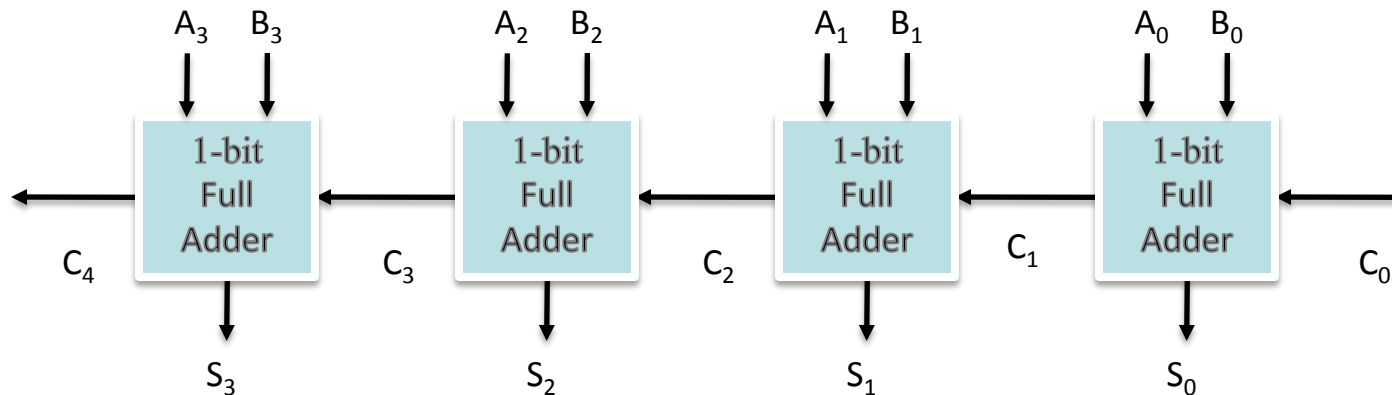
- Full Adder



Input A	Input B	Carry in	Carry Out	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

# Four Bit Adder

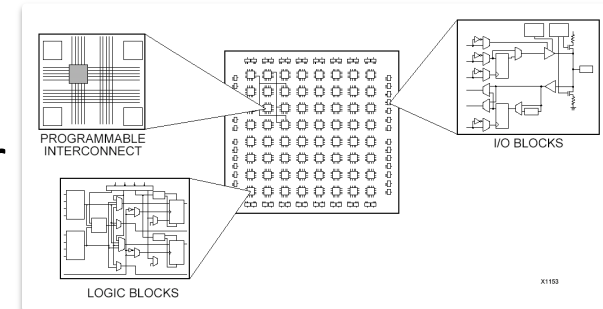
- Cascade Four Single Bit Full Adders



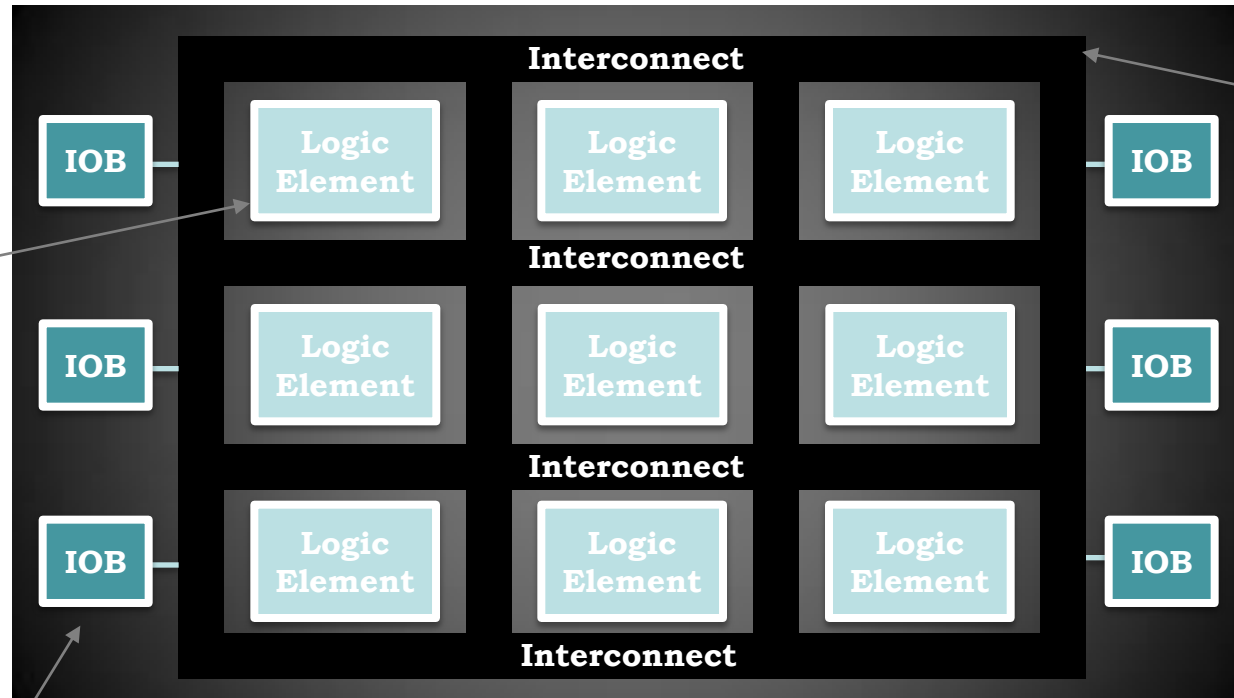
Quiz: Propagation Delay ?

# What is an FPGA?

- **What it is**
  - Field-programmable gate array
  - A silicon chip with unconnected gates
  - User can define and re-define functionality
- **How it works**
  - Define behavior in software
  - Compile and download to the hardware
  - Hardware implementation of code
- **When it is used**
  - Custom hardware or ICs, replacement for ASICs
  - Reconfiguration required after deployment



# FPGA Hardware Fabric



Logic Elements are basic building blocks of an FPGA and can be programmed to carry out different function as required by the design

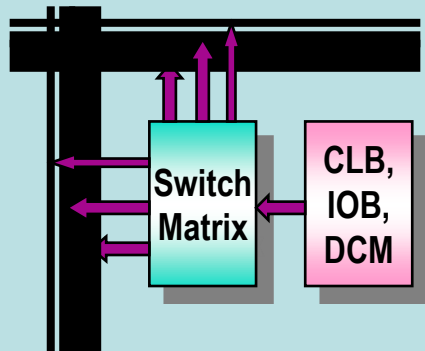
Interconnects wire different logic cells together to form more complex design blocks

Input/Output Block connect internal FPGA architecture to the external design via interfacing pins

Note: Precise architecture of an FPGA varies from manufacturers to manufacturers.

Every manufacturer has a different version of a specific FPGA; basically increasing the number of logic cells.

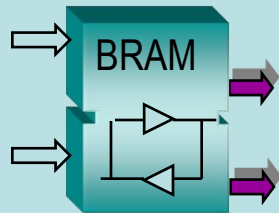
## Active Interconnect™



- Fully buffered
- Fast, predictable

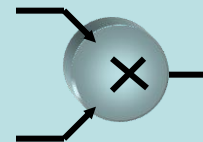
## Block RAM

- 18KBit True Dual-Port
- Up to 3 Mbits per device

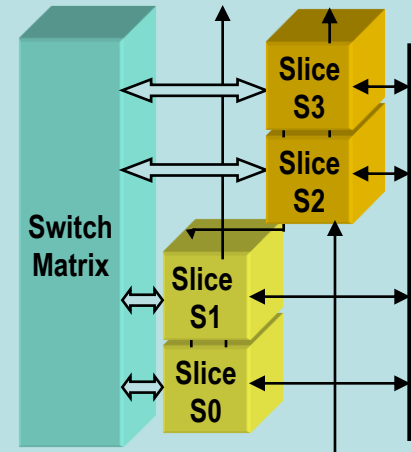


## Multipliers

- 18b x 18b multiplier
- 200+ MHz pipelined



## Powerful CLB



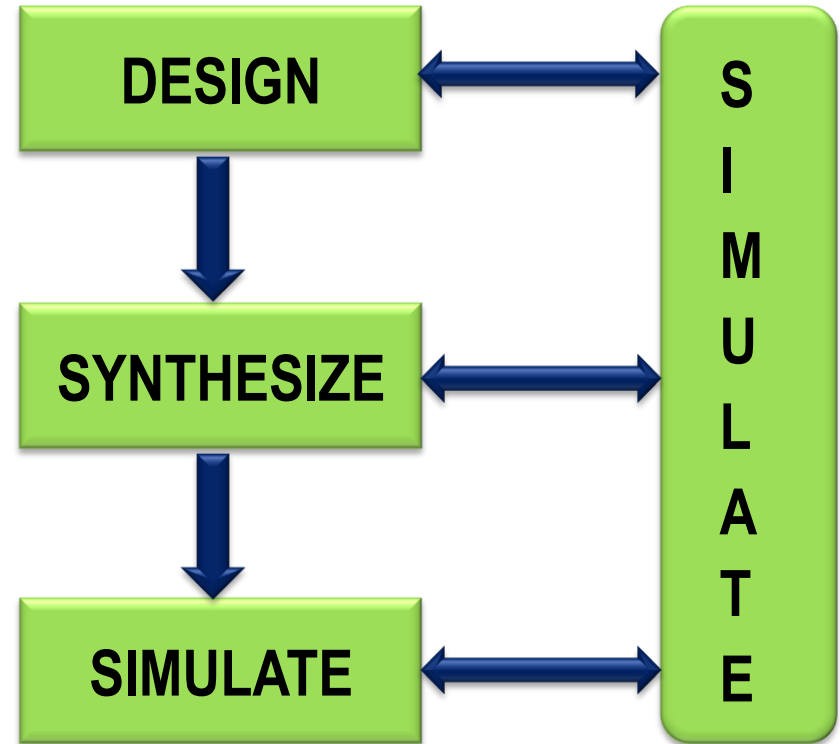
- 8 LUTs
- 128b distributed RAM
- Wide input functions (32:1)
- Support for slice-based multipliers

## Typical FPGA Architecture (Xilinx Virtex II & Virtex II Pro)

# Development Stages

Four development stages

- Design
- Simulate
- Synthesize
- Simulate



# Step #1: Design

- Schematic capture
  - Graphical depiction
  - Easy to understand
  - Vendor specific
  - Ex: ViewDraw, Ease
- Hardware Description languages (HDL)
  - Text based – “Firmware”
  - Generic or vendor specific
  - Ex: VHDL, Verilog

# HDL Styles

- Structural
  - Software equivalent of schematic capture
  - Uses vendor specific components
  - Repeat design process for different vendors
- Behavioral
  - Describe digital functions in generic terms
  - Vendor independent

# Step #2: Simulate

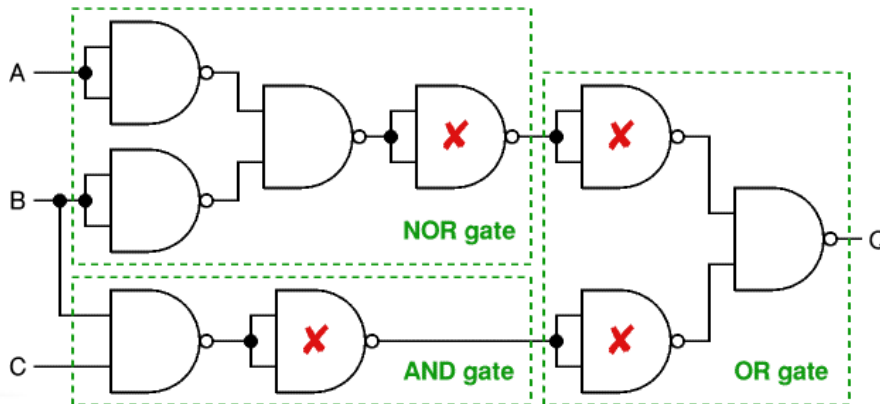
- Verification of code
- Simulate after each step
  - Register transfer (RTL) level
  - Functional
  - Gate level
- Testbenches – Apply stimulus
  - Automatic
  - Manual
  - Ex: ModelSim, Riviera

# Testbenches

- RTL Level
  - Verify logic of code
  - No timing
- Functional level
  - Occurs after synthesis
  - Verify intactness of design
- Gate level
  - Occurs after implementation
  - Verify timing

# Step #3: Synthesis


- Reduces and optimizes design
  - Creating structural elements
  - Optimizing
  - Mapping



# Step #4: Implementation

- Final stage
- Place and route
  - Automatic or manual pin assignment
  - Uses constraint file
- 3 steps
  - Translate
  - Fit
  - Generate program file

# VHDL Basics

- VHDL  $\longrightarrow$  VHSIC Hardware Description Language  
  
Very High Speed Integrated Circuit

- Industry Standard for Description, Modeling and Synthesis of Digital Circuits and Systems

# VHDL Framework & Syntax

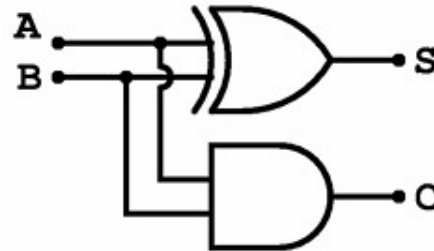
- VHDL Description of Logic Blocks is split into
  - **Entities**
    - Defines the **Inputs & Outputs** of a design
    - Similar to a **declaration** of a function in C, C++
  - **Architecture**
    - **Describes** the **logic** behind the entity
    - Similar to a **description** of a function in C, C++

# VHDL Example: 1-Bit Adder

```
library ieee;  
use ieee.std_logic_1164.all;
```

```
ENTITY Single_Adder IS PORT
```

```
(  
    A, B: IN std_logic ;  
    S, C: OUT std_logic  
);  
END Single_Adder;
```



```
ARCHITECTURE Architecture_Single OF Single_Adder IS
```

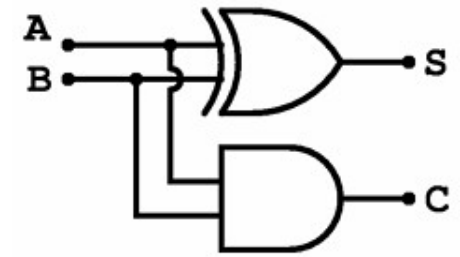
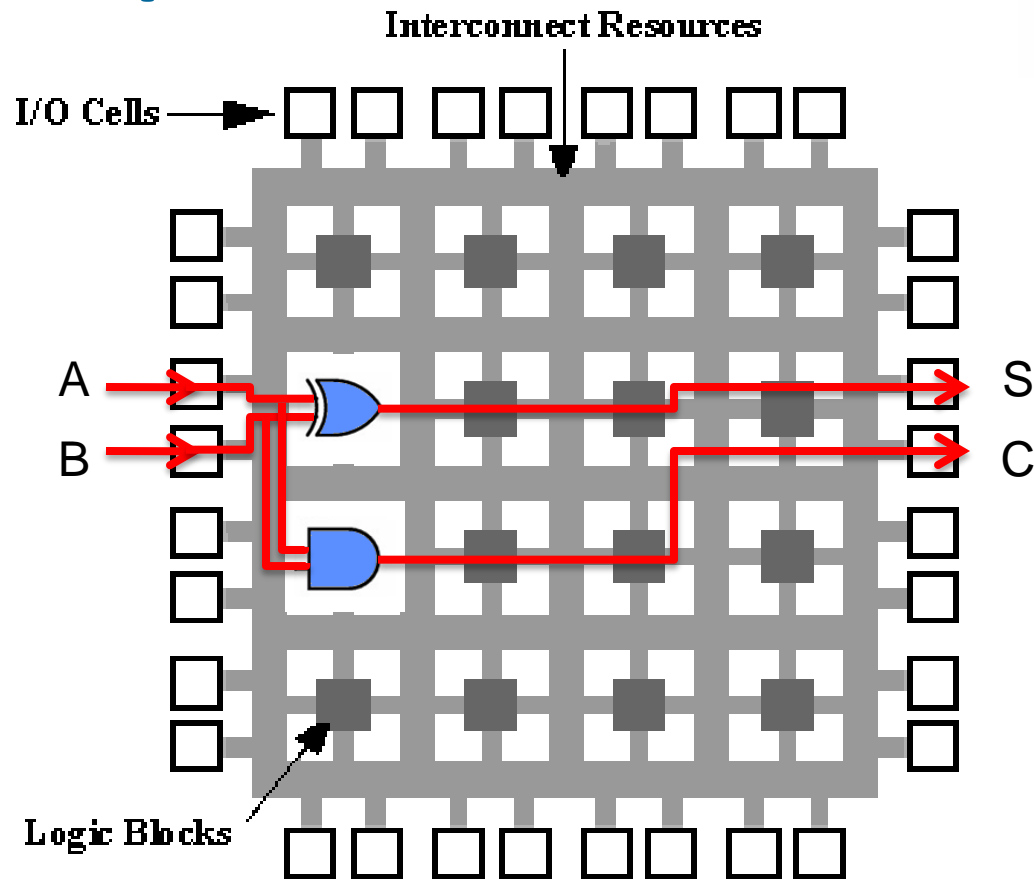
```
BEGIN
```

```
    S <= A xor B;
```

```
    C <= A and B;
```

```
END Architecture_Single;
```

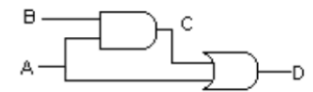
# FPGA Implementation



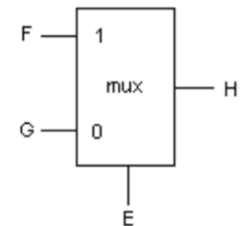
# HDL – Statement Types

- Concurrent statements act at the same time
  - Think of a schematic
- A Process contains sequential statements
  - Like software
- Parallel and Sequential
  - Processes run in parallel and interact concurrently

```
architecture RTL of Execution_Example is
signal A,B,C,D,E,F,G,H : std_logic;
begin
--concurrent statements
C <= A and B;
D <= A or C;
```



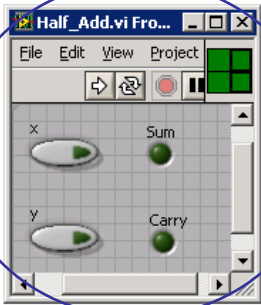
```
Proc1 : process (E,F,G)
begin
-- sequential statements
if (E = '1') then
H <= F;
else
H <= G;
end if;
end process Proc1;
end RTL;
```



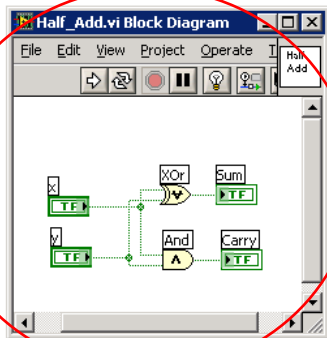
```
end process Proc3;
--concurrent statements
end RTL;
```

# Language Basics - Entity Architecture Pair

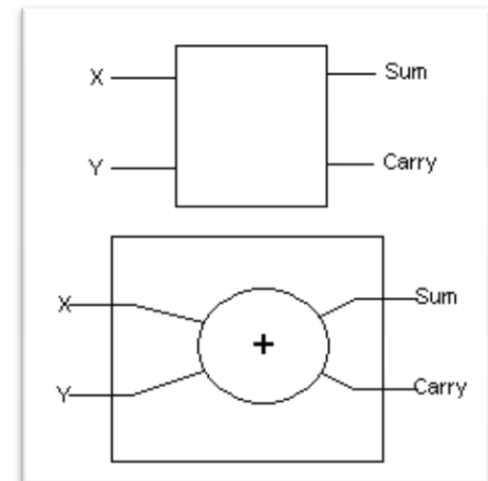
- Entity is like Front Panel and connector Pane
- Architecture is like block diagram



```
entity Half_Add is  
  port ( X, Y : in std_logic;  
         Sum, Carry : out std_logic);  
end Half_Add;
```

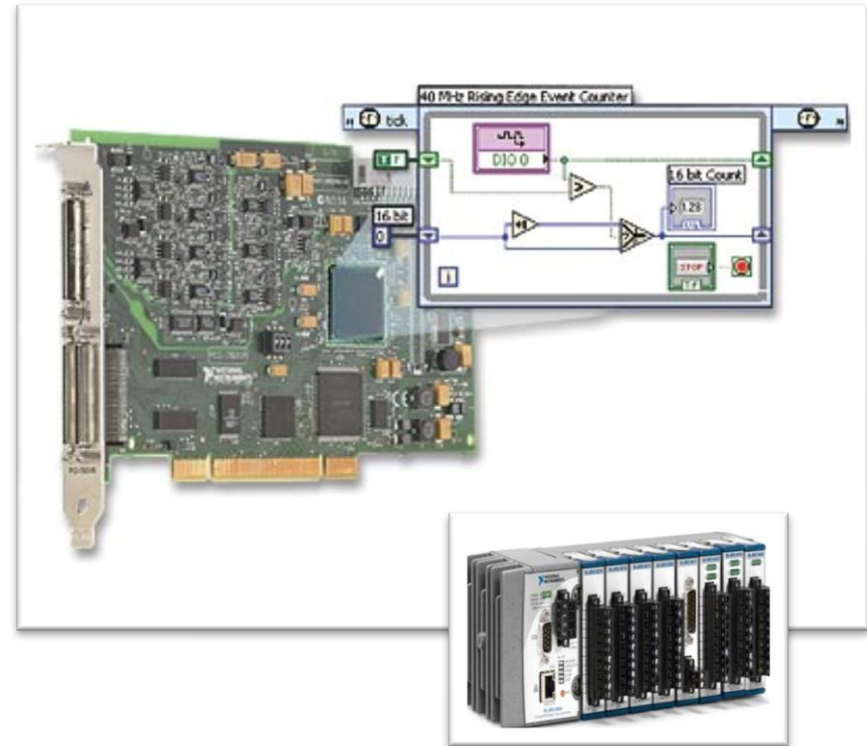


```
architecture RTL of Half_Add is  
begin  
  Sum <= X xor Y;  
  Carry <= X and Y;  
end RTL;
```

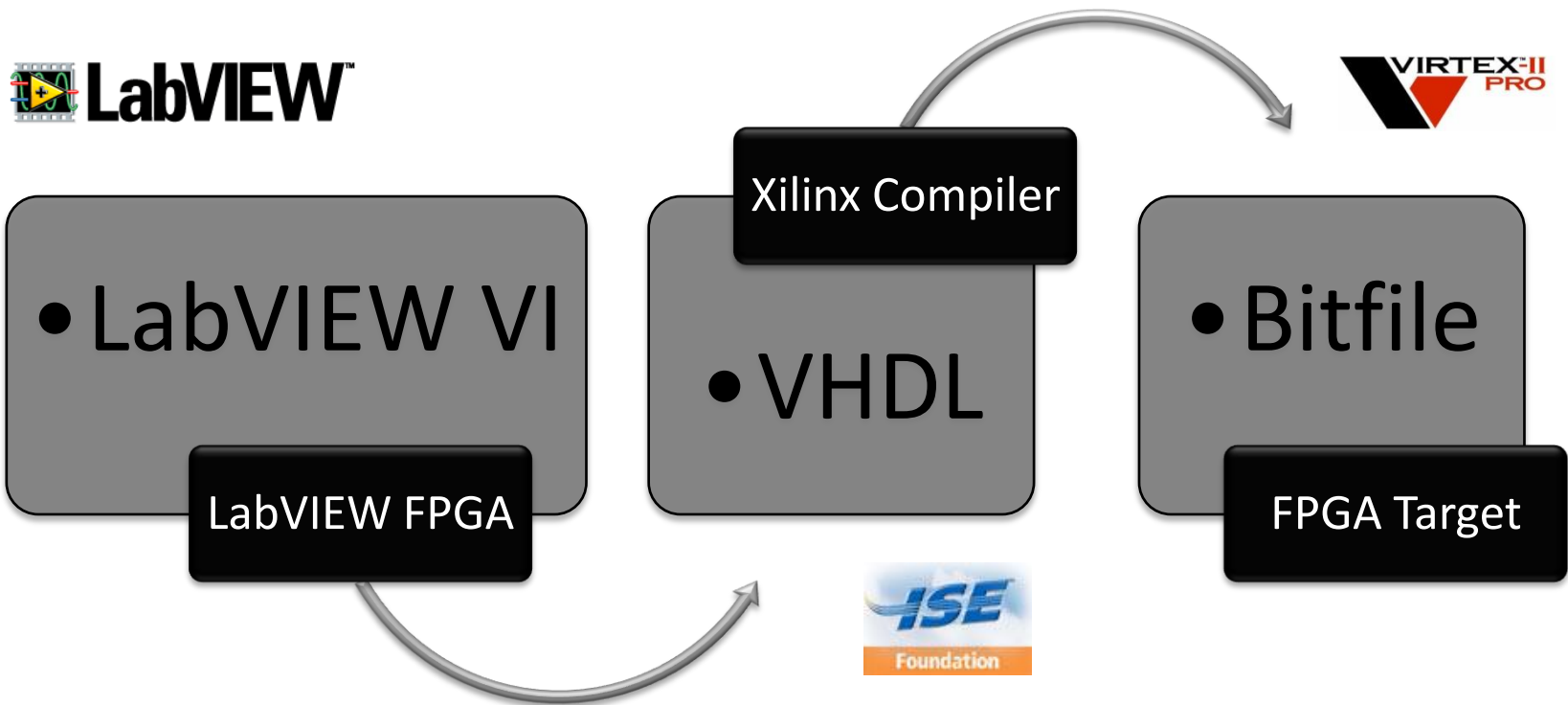


# LabVIEW FPGA Module

- Software for developing VIs for FPGA target
- VIs for host PC interaction with FPGA target
- Target LabVIEW FPGA Enabled Hardware
  - Plug-In Reconfigurable I/O (RIO) boards
  - CompactRIO Modular Reconfigurable I/O System
  - Compact Vision System

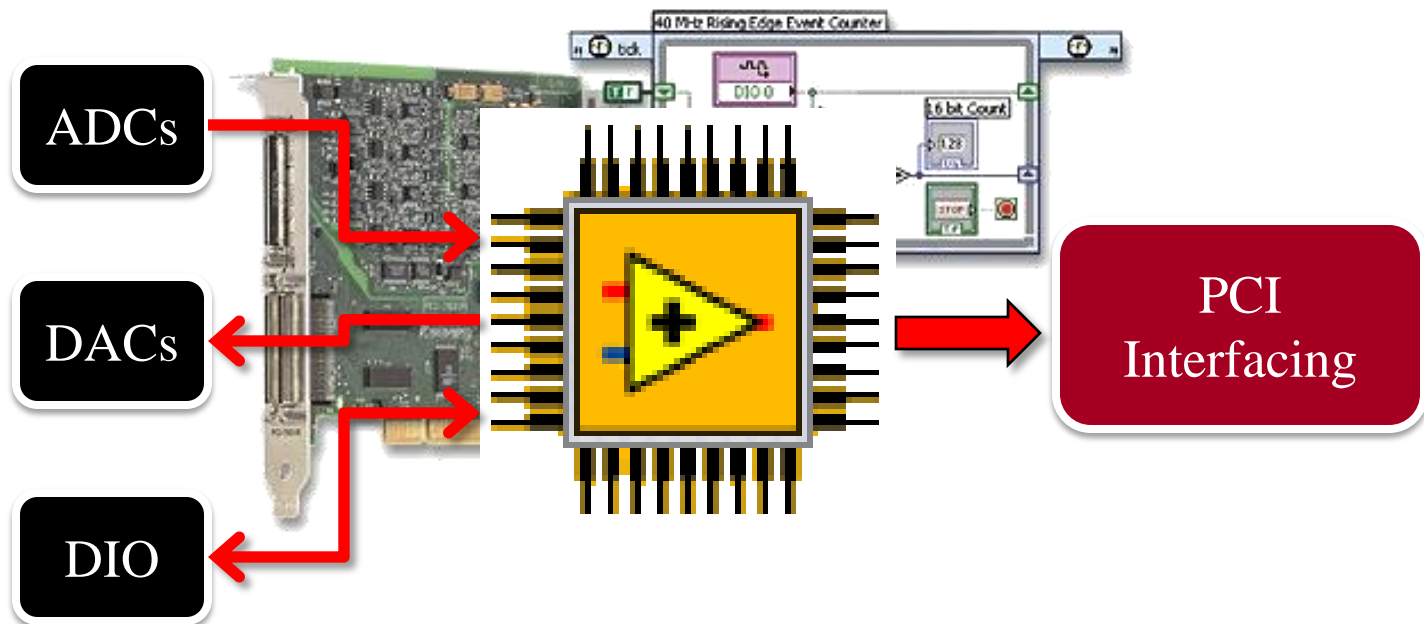


# LabVIEW FPGA Tool Chain



# Advantages of FPGA Based Systems

Example: R-Series



Similar to M-series, DAQ-STC replaced with an FPGA

# VHDL Example: 4-Bit Adder

```

library ieee;
use ieee.std_logic_1164.all;

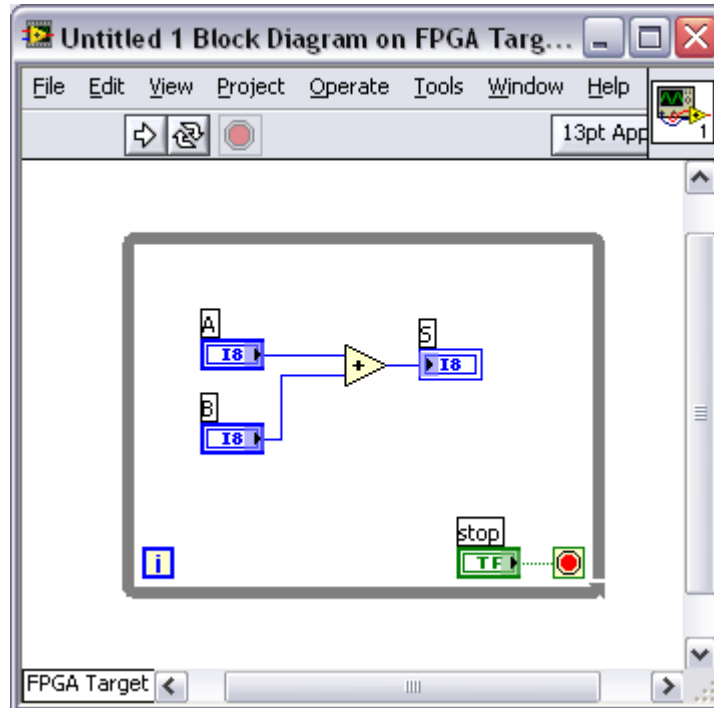
ENTITY Four_Bit_Adder IS PORT
(
  A, B: IN std_logic_vector (0 downto 3) ;
  Cin: IN std_logic;
  S: OUT std_logic_vector(0 downto 3);
  Cout: OUT std_logic
);
END Four_Bit_Adder;

ARCHITECTURE Architecture_Four OF Four_Bit_Adder IS
  COMPONENT Single_Adder PORT
  (
    A, B, Cin: IN std_logic ;
    S, C: OUT std_logic);
  END COMPONENT;
  SIGNAL Cint: std_logic_vector (0 downto 2)
  BEGIN
    Add1: Single_Adder port map (A(0), B(0), Cin, S(0), Cint(0));
    Add2: Single_Adder port map (A(1), B(1), Cint(0), S(1), Cint(1));
    Add3: Single_Adder port map (A(2), B(2), Cint(1), S(2), Cint(2));
    Add4: Single_Adder port map (A(3), B(3), Cint(2), S(3), Cout);
  END Architecture_Four;

```



# LabVIEW FPGA Equivalent





# Summary

- 4 Main steps in putting together an FPGA
- Good for custom circuits and reliable architectures
- Tools abstracting the complexity available