MAPLD

Accelerating FPGA Designs and Design Work: Implementing Faster Designs Faster

Bryan Penner
Xilinx FAE – Arizona and New Mexico

XILINX®

Agenda

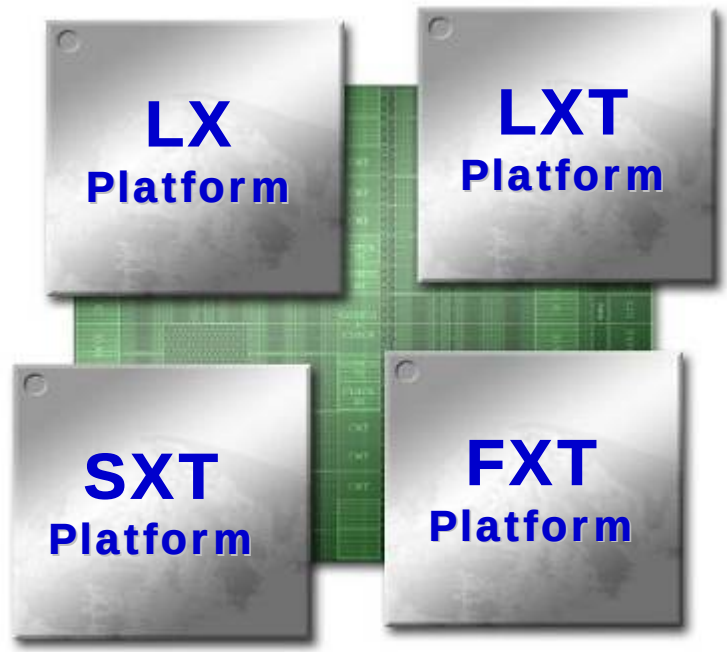
- Understanding the Virtex5 FPGA Architecture
- A look at how coding affects performance
- Software tools that can help increase performance and reduce design time

Agenda

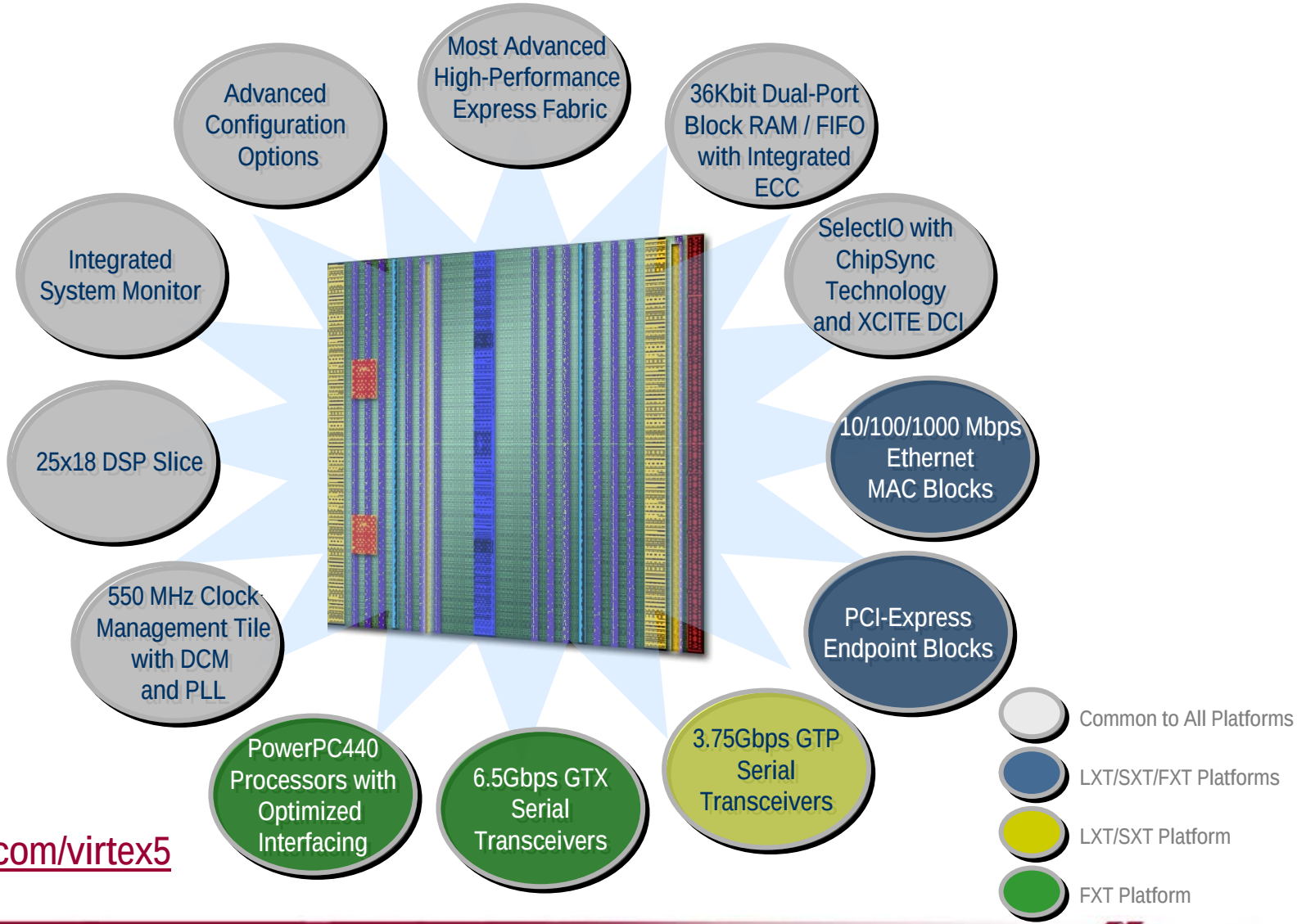
- Understanding the Virtex5 FPGA Architecture
- A look at how coding affects performance
- Software tools that can help increase performance and reduce design time

The Virtex-5 Family

- Family contains 4 platforms
  - LX
    - High-performance logic
  - LXT
    - High-performance logic with lowest power serial connectivity
  - SXT
    - Extensive signal processing with lowest power serial connectivity
  - FXT
    - Embedded-oriented with highest performance microprocessor and serial connectivity



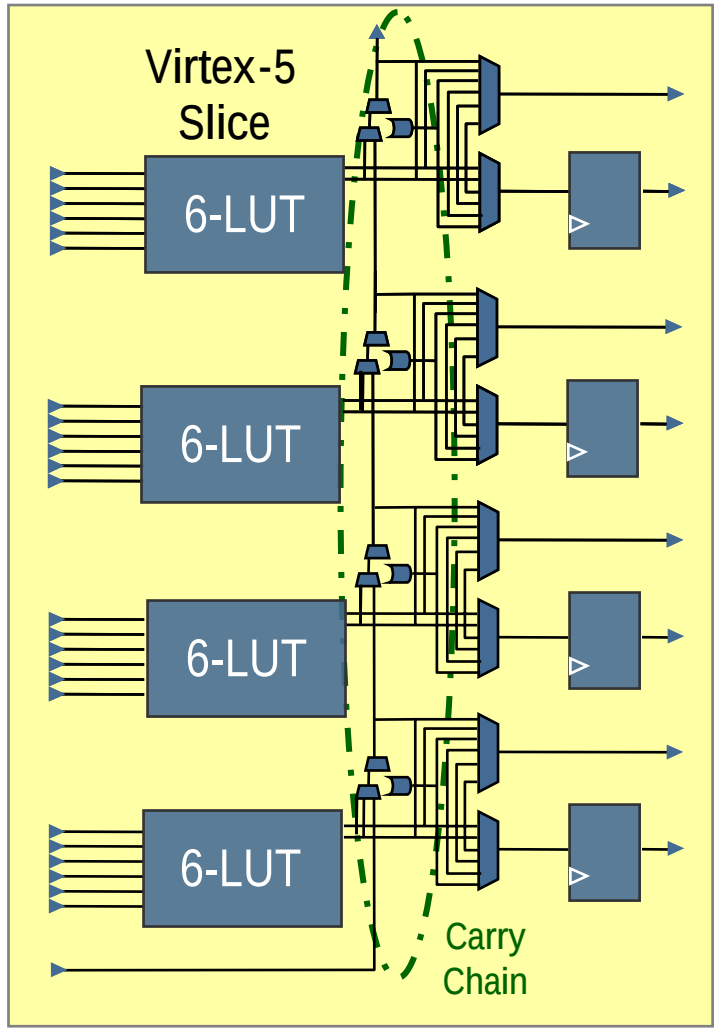
Continuing the Drive for Innovation



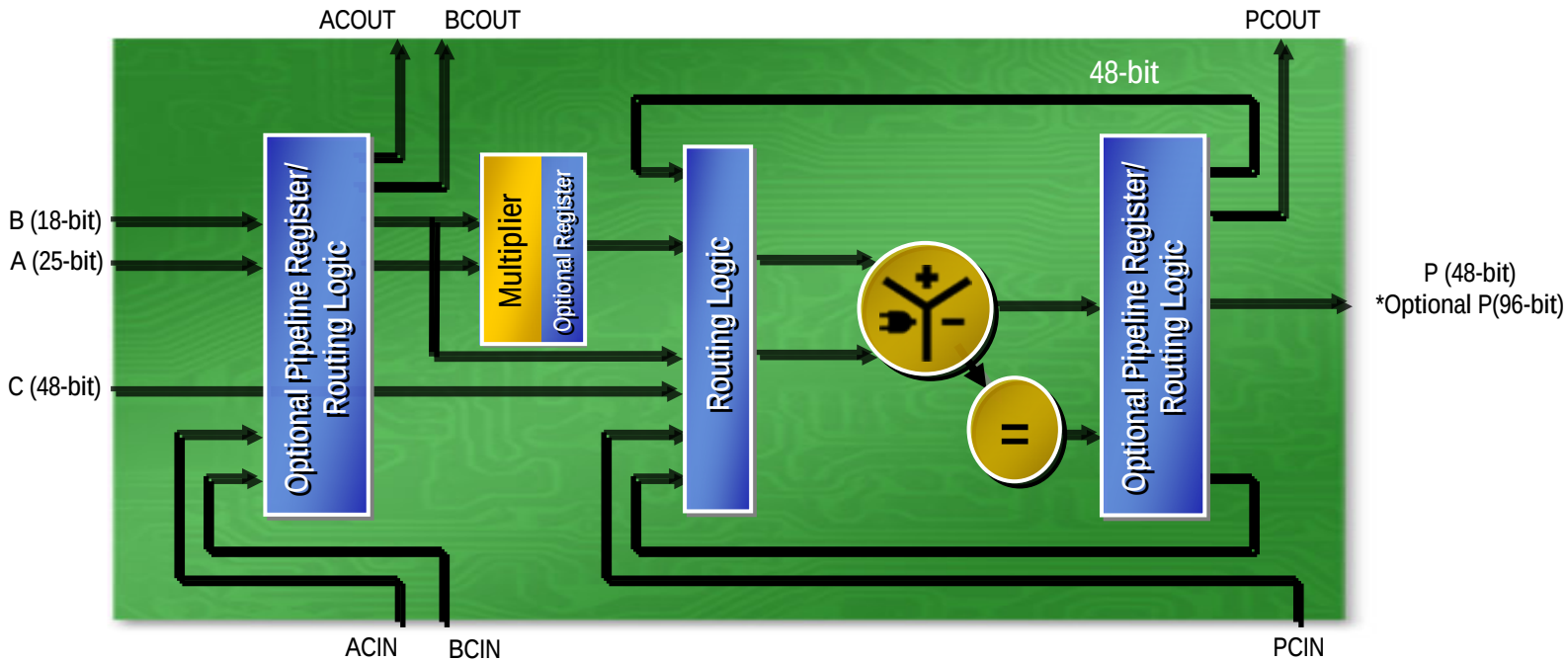
www.xilinx.com/virtex5

Virtex-5 Slice with 6-Input LUTs

- 6-Input LUT with six independent inputs
  - Four 6-input LUTs per slice
  - Two outputs per LUT
- Fast Carry Chain
  - addition
  - subtraction
- High Performance Flip Flops
  - Synchronous or asynchronous active high reset, set and clock enable
- 6-Input LUT configured as
  - Any 6-input logic function
  - 64 bit Distributed RAM
  - 32 bit Shift Register
- More efficient interconnect



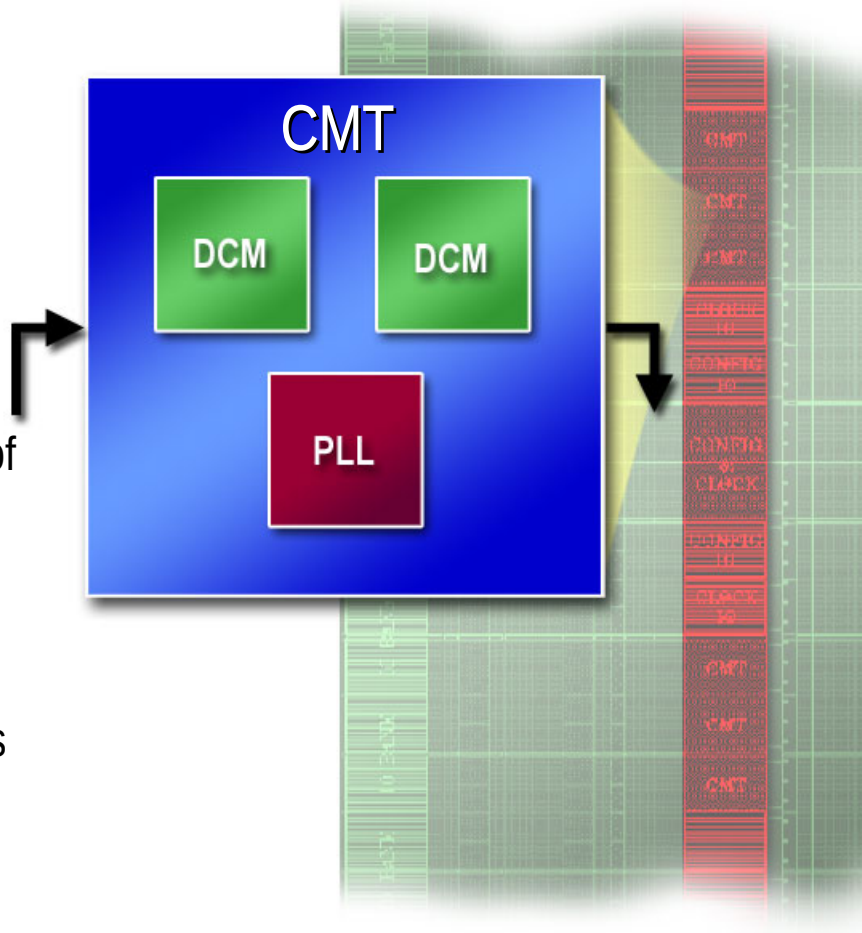
Virtex-5 DSP48E For Efficient DSP



* 96-bit output using MACC extension mode (uses 2 DSP48E slices)

Virtex-5 Clock Management Tile

- Up to 6 CMTs per device
  - Each with 2 DCMs and 1 PLL
  - No external PWR/GND pins
- DCM
  - Operate from 19 MHz – 550 MHz
  - Remove clock insertion delay
    - "Zero delay clock buffer"
  - Dynamically phase shift clocks in increments of period/256 or with direct delay line control
- PLL
  - Operate from 19 MHz – 550 MHz
  - Reduces internal clock jitter
  - Supports higher jitter on reference clock inputs
  - Remove clock insertion delay
    - "Zero delay clock buffer"
  - Synthesize $F_{out} = F_{in} * M/(D*O)$



Agenda

- Understanding the Virtex5 FPGA Architecture
- A look at how coding affects performance
- Software tools that can help increase performance and reduce design time

Intro

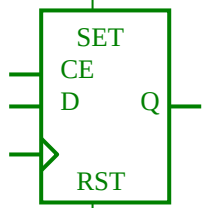
- There is not a single way to create a design
  - Different coding styles, synthesis / implementation tool options will lead to different results
    - And no one formula will work best in all cases
- There are however guidelines that can generally lead to improved *performance*, *area* and *power*
  - I am not telling you how to code your design
    - I am trying to relay the ramifications and drawbacks of some typical coding decisions

Flip-Flops

Local set (Q=1) can be asynchronous or synchronous.

Clock Enable qualifies that the clock edge should be used to store data.

Data input

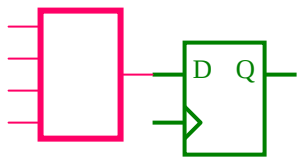


Initial value of Q output will relate to the controls used.

Data stored synchronously on a positive or negative clock edge.

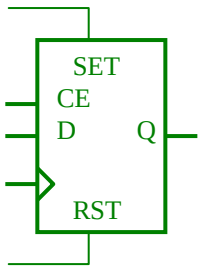
Local reset (Q=0) can be asynchronous or synchronous.

The D-input naturally connects to the output of the LUT and leads to best density and highest performance.



Control Priority

- Control inputs to the flip-flops have a predictable priority



Synchronous reset has highest priority

Synchronous set has second priority

Clock Enable has lowest priority

FDRSE FDCPE - flip-flop with asynchronous clear and preset.

- Write HDL code which is sympathetic to the control priorities
- Do not mix synchronous and asynchronous controls as these are not be supported

Flip-Flop Controls

- Eight bit data register with reset (global reset?)
- Reset forces output to “00000000”.
- Synchronously set to “11111111”
- Input value captured when enable is high

```
signal reg_data: std_logic_vector(7 downto 0);
```

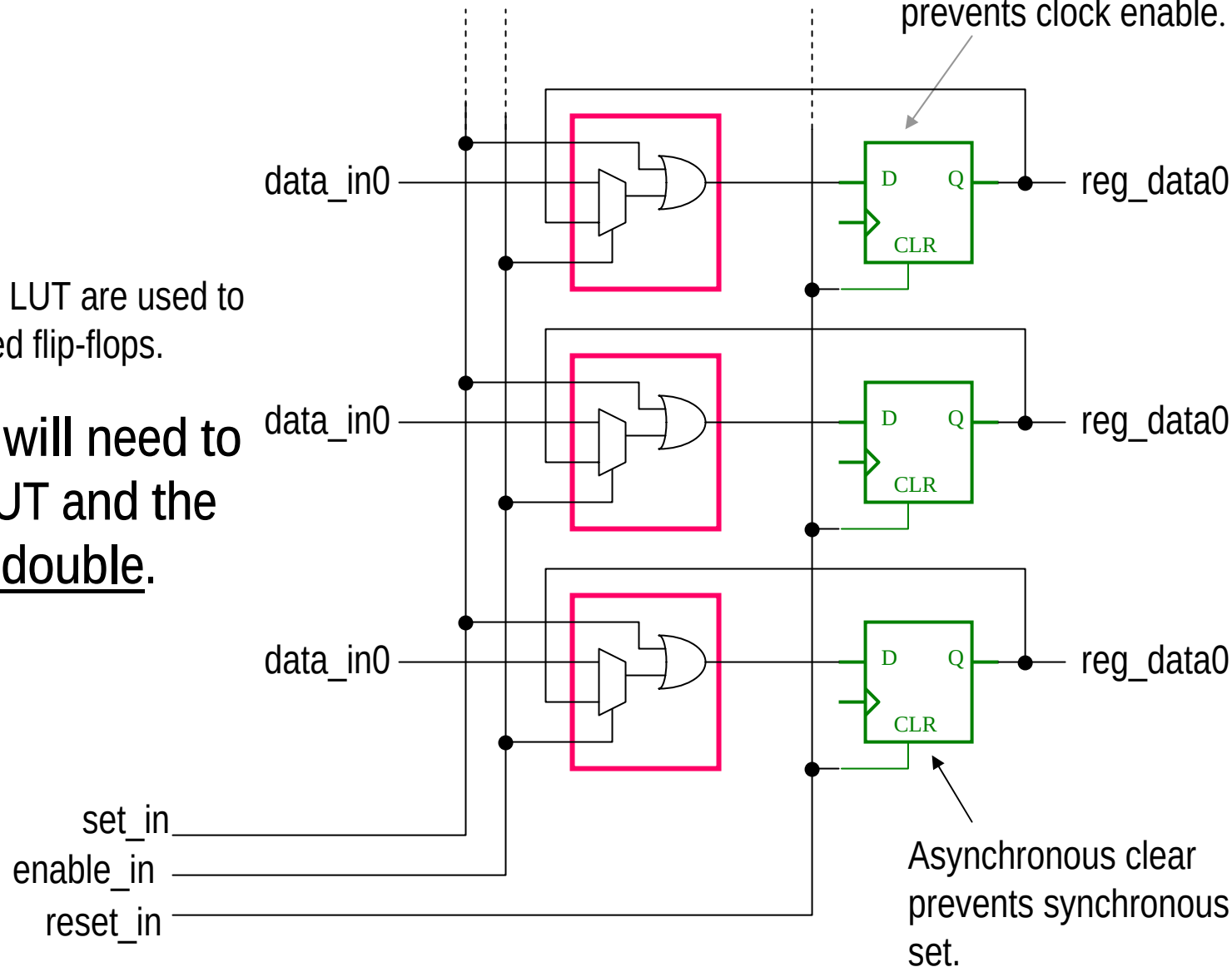
```
byte_register: process (clk, reset_in)
 begin
   if reset_in ='1' then
       reg_data <= "00000000";
     elsif clk'event and clk='1' then
       if set_in ='1' then
           reg_data <= "11111111";
         elsif enable_in='1' then
           reg_data <= data_in;
       end if;
   end if;
 end process;
```

- Code looks reasonable. Might assume it will require 8 FFs to implement

Flip-Flop Controls

All 4 inputs of each LUT are used to emulate the required flip-flops.

Design logic will need to use other LUT and the cost will double.



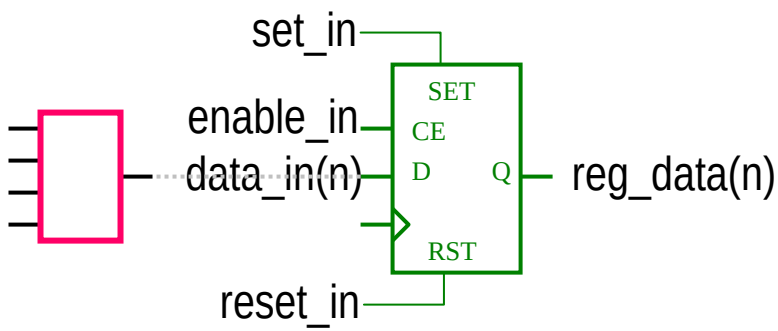
Flip-Flop Controls

Improvement : Make the reset a synchronous control.

VHDL

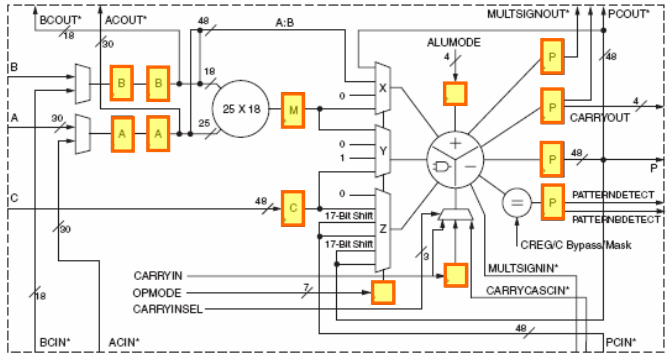
```
byte_register : process (clk)
 begin
   if clk'event and clk='1' then
     if reset_in ='1' then
         reg_data <= "00000000";
       elsif set_in ='1' then
           reg_data <= "11111111";
         elsif enable_in='1' then
             reg_data <= data_in;
     end if;
   end if;
 end process;
```

Result: 8 flip flops of type FDRSE



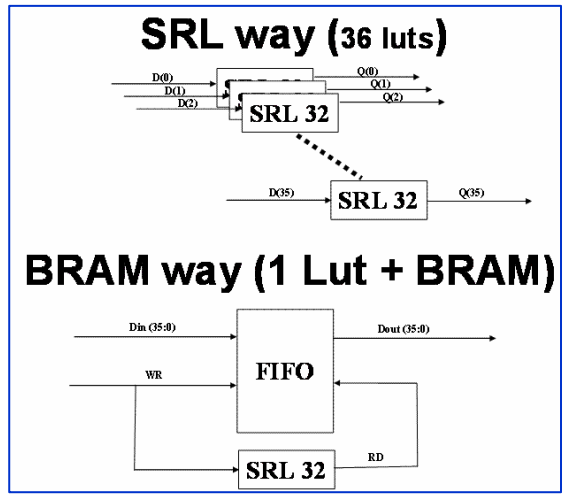
Synchronous Resets

- Use of the DSP48E only possible if synchronous resets are used
- Asynchronous resets will result in a significantly slower Fmax and under utilization of this valuable resource



Each DSP48E has ~250 registers, all with synchronous reset

- BlockRAMs get minimum clock-to-out by using the output registers
  - Output registers only have synchronous resets
- Unused BlockRAMs can be used for alternative purposes
  - ROMs, Large Look Up Tables, Complex logic, State-Machines, Large Shift Registers, Dynamic Updating Logic
  - Cannot be used if design uses asynchronous resets

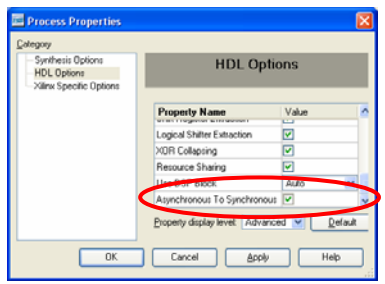


How to Change to Synchronous Resets

- It is suggested that all new code should use synchronous resets when a reset is necessary
- For existing code, you have 3 choices
  - Leave alone
    - Acknowledge the possible drawbacks of asynchronous resets
  - Use synthesis switch

Synplicity:
`syn_clean_reset`

XST:
`-async_to_sync YES`



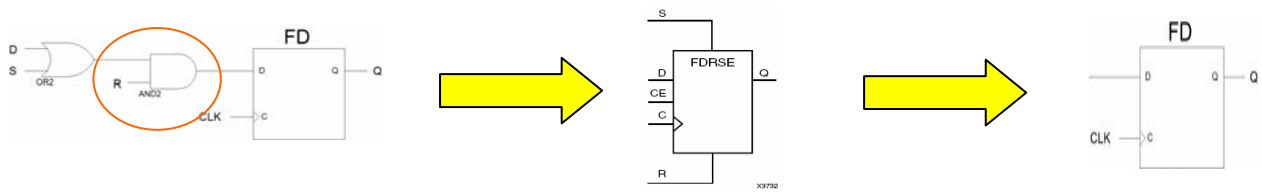
Not the same as changing to synchronous reset but can help

  - Manually change the asynchronous reset to a synchronous

What's Better than Synchronous Resets?

Why No Resets at All?

More Free Logic Even Fewer Control Signals



Async Reset

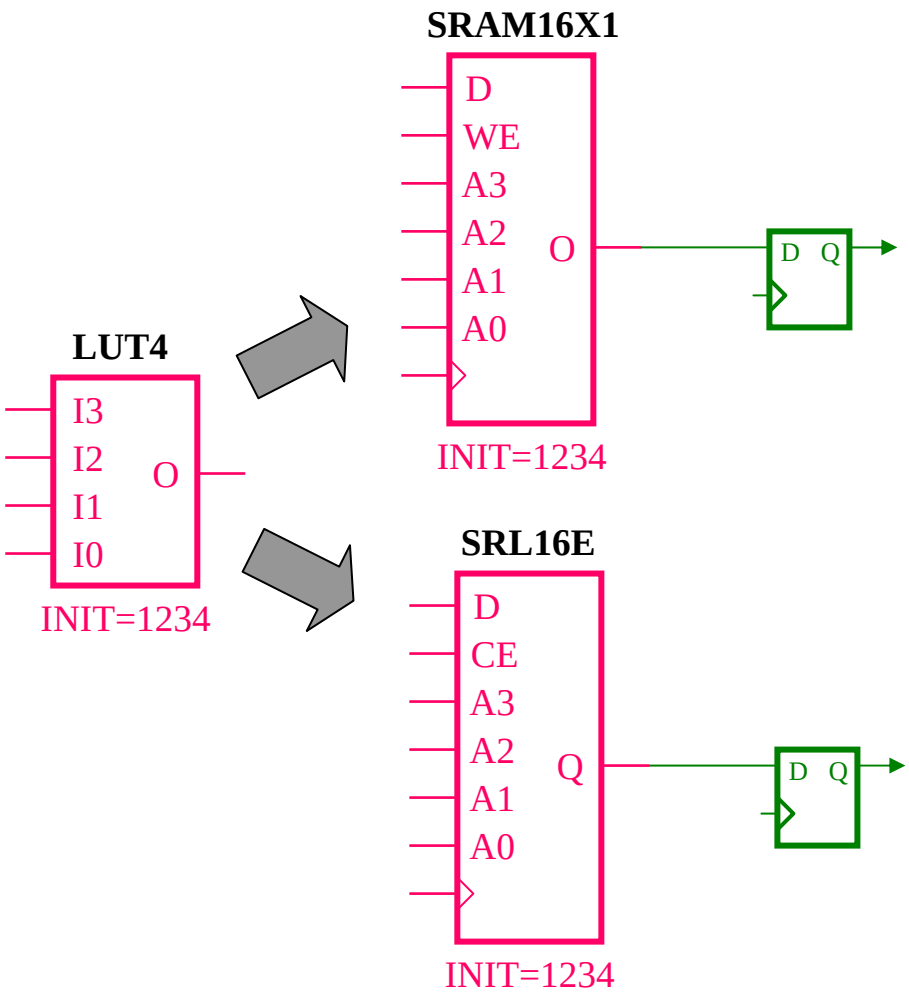
Sync Reset

No Reset

- Using synchronous resets frees up additional logic
  - Potentially, a “free” AND and/or OR gate can be realized for every FF in the design
- Greater register packing within Slices may be realized
  - Greater flexibility for registers packing with fewer control signals

Why No Resets at All?

No reset on LUTRAM

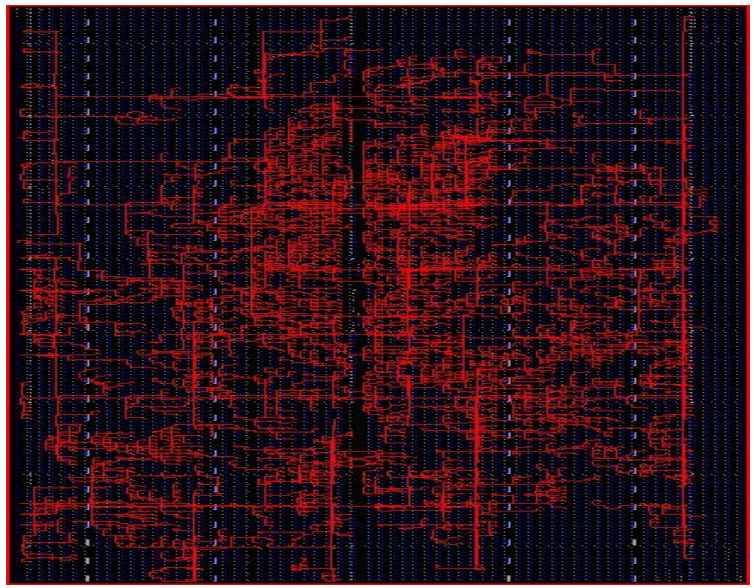


- Coding a reset when describing a RAM or shift register will prevent the use of LUTRAM
- The DistRAM is synchronously written, but asynchronously read.
- Follow the RAM with the dedicated FF to make a synchronous read and improve performance.

- The dedicated FF has a faster clock to out time than the SRL16E LUT.
- Synthesis should place the last register in shift chain in the FF.
- The initial contents of the LUTRAM can be specified or zero will be the default value.

Why No Resets at All?

Routing Congestion

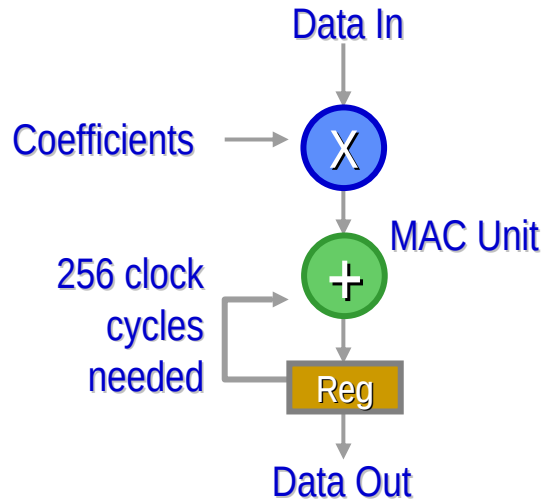


- Routing can be considered one of the more valuable resources
- Resets compete for the same resources as the rest of the active signals of the design
  - Including the critical paths
- Designs without resets have fewer timing paths
  - By an average of 18% fewer timing paths
- Results in less runtime

FPGAs Enable Massively Parallel DSP

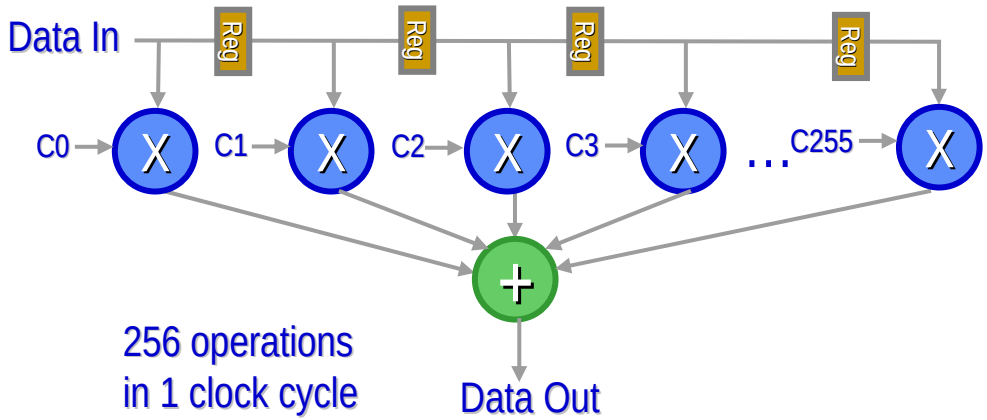
Example 256 TAP Filter Implementation

Programmable DSP - Sequential



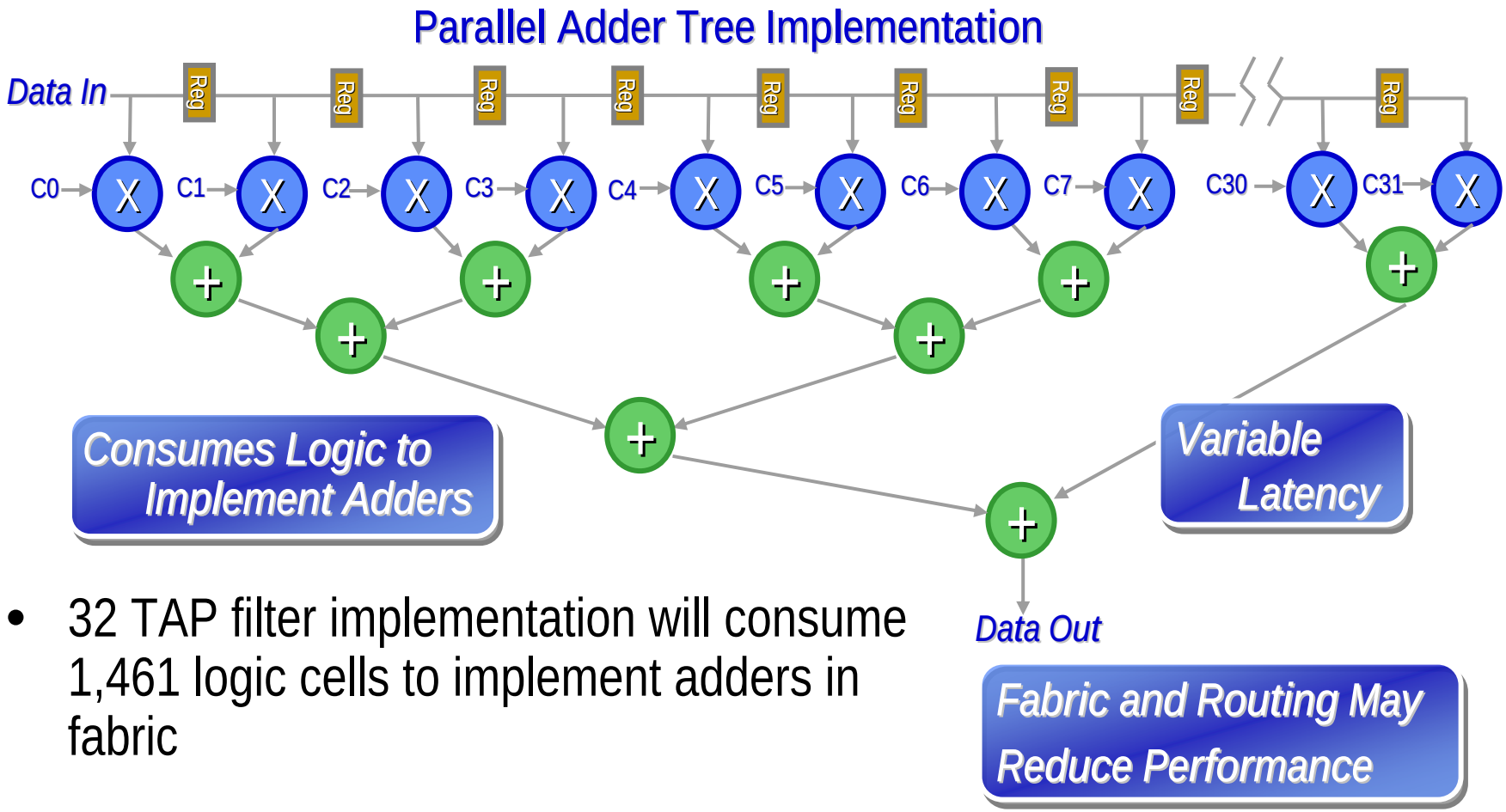
$$\frac{1\ GHz}{256\ clock\ cycles} = 4\ MSPS$$

FPGA - Fully Parallel Implementation



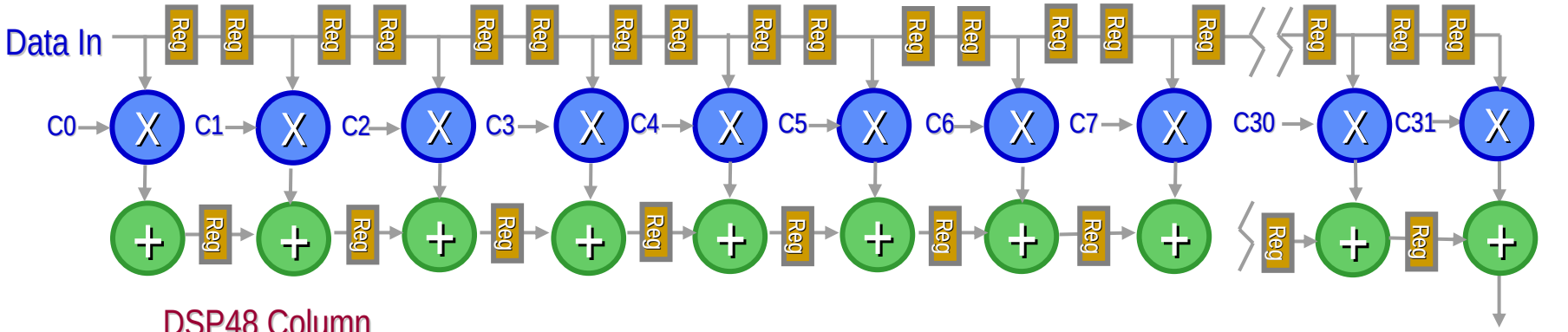
$$\frac{500\ MHz}{1\ clock\ cycle} = 500\ MSPS$$

Parallel Adder Tree Implementation Consumes FPGA resources

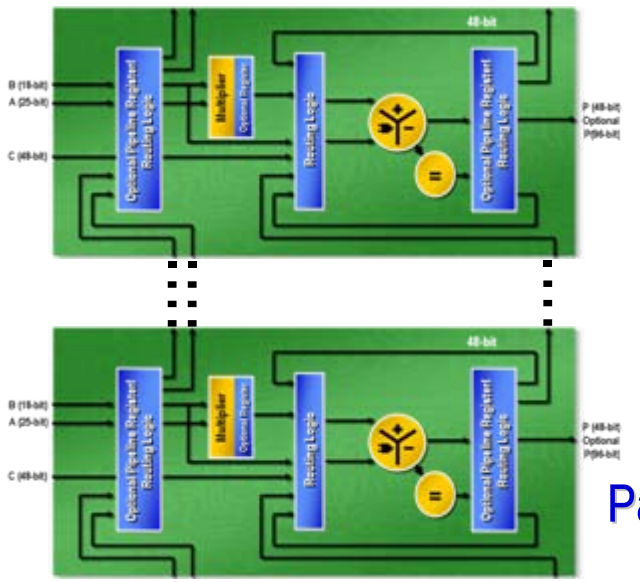


- 32 TAP filter implementation will consume 1,461 logic cells to implement adders in fabric

Parallel Implementation Consumes Zero Logic Resources



DSP48 Column



- 32 TAP filter implementation using 32 XtremeDSP Slices
- Guaranteed 550 MHz operation
- HDL coding examples in Virtex-5 FPGA XtremeDSP Design Considerations User Guide

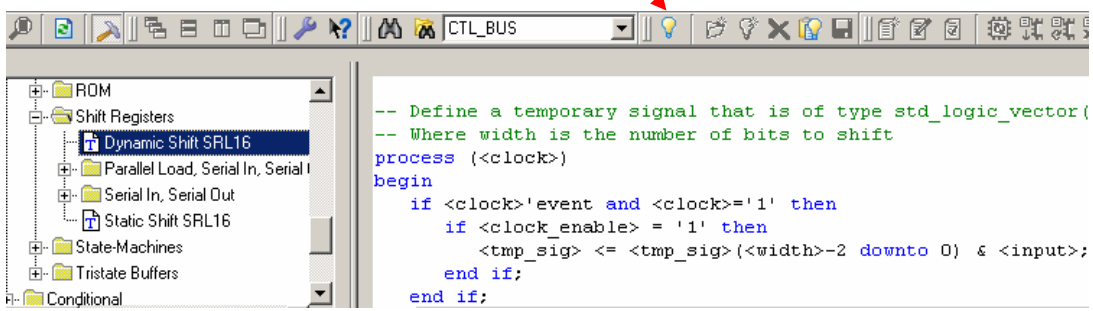
Parallel Adder Cascade Implementation in DSP48 Column

Pipelining

- Pipelining cannot be an afterthought
  - Adding pipeline registers “later” is not easy
  - Number and placement of registers need to be considered during initial coding
- Too little pipelining will result in under-performing designs
- Maximum performance seen when…
  - There are 6 inputs to a logic function
    - This is different than previous architectures due to the 6-LUT
  - Extra caution is taken around Multipliers and RAMs

Where to find more information

- WP231 – HDL Coding Practices to Accelerate Design Performance
- WP272 – Get Smart About Reset: Think Local Not Global
- WP271 – Saving Costs with the SRL16E
- WP333 – FIFOs in Virtex-5 FPGAs
- WP284 – Advantages of Virtex-5 FPGA 6-Input LUT Architecture
- WP275 – Get your Priorities Right – Make Your Design up to 50% Smaller
- WP248 – Retargeting Guidelines for Virtex-5 FPGAs
- WP245 – Achieving Higher System Performance with the Virtex-5 Family of FPGAs
- Synthesis and Simulation Design Guide in Software Manual.
- Coding Examples in the Language Template

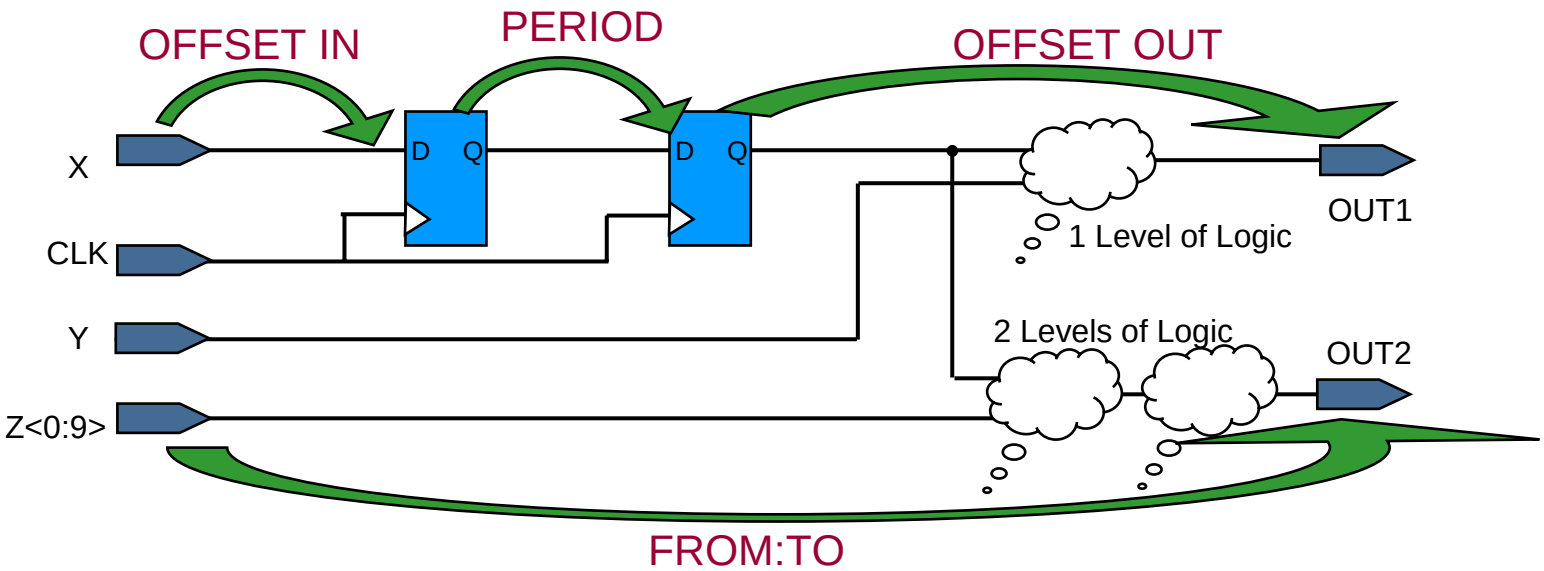


Agenda

- Understanding the Virtex5 FPGA Architecture
- A look at how coding affects performance
- Software tools that can help increase performance and reduce design time

Timing Constraints

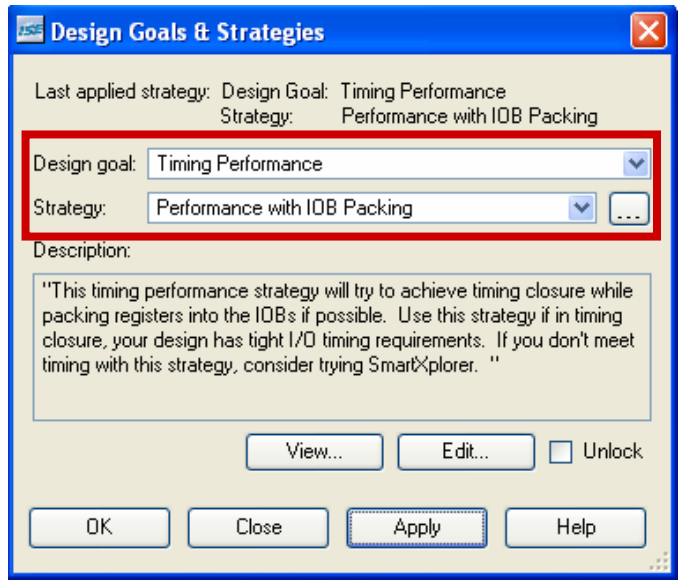
- All designs should have timing constraints for IO, clocks and multi-cycle paths
- Implementation tools are timing driven
  - Without timing constraints implementation only concern is runtime
- Synthesis tools are also timing driven
  - Synthesis will make logic decisions based on timing constraints
- See the Constraints Guide in Software Manual
- Language Template in Project Navigator has constraint examples



Strategy-Based Implementation

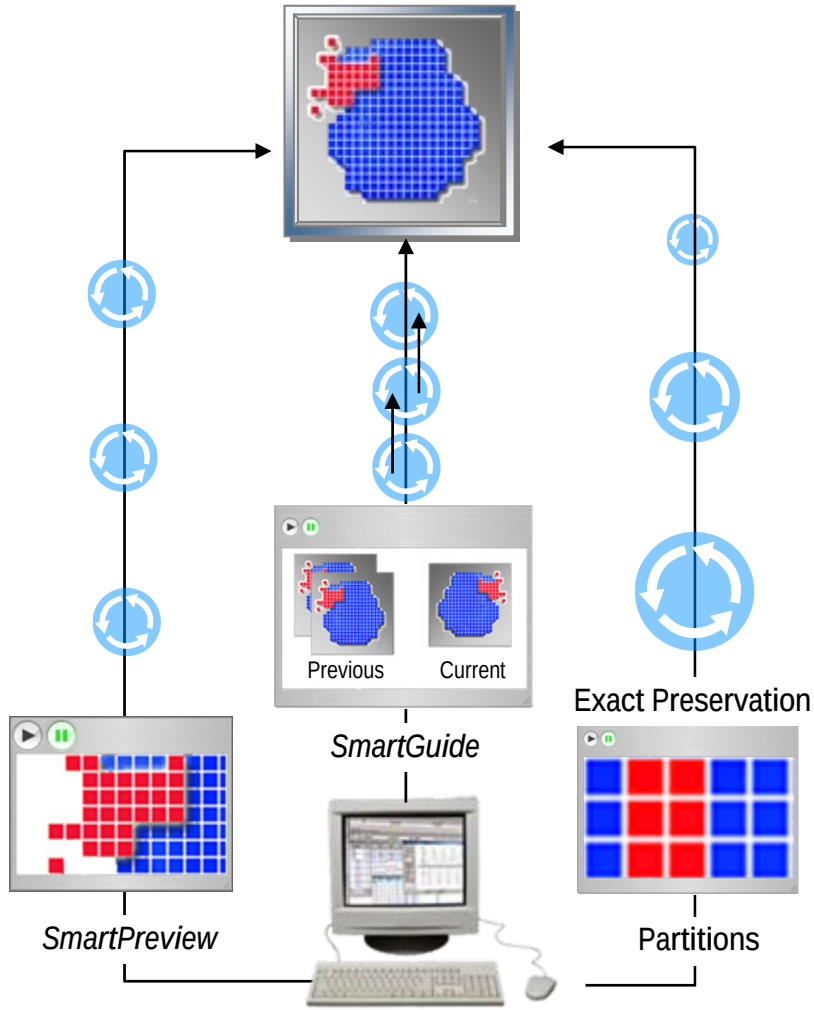
- Software switches can impact performance, area and power
- Automatically identifies optimal implementation algorithm based on design goals
  - Balanced: (Default) Delivers balance of performance and runtime
  - Timing Performance: Delivers optimal performance
  - Minimum Runtime: Focuses on minimizing runtime
  - Area Reduction: Slice Reduction with minimal impact to performance
  - Power Optimization: Minimizes dynamic power with minimal impact to performance

Set the Goal instead of multiple implementation settings



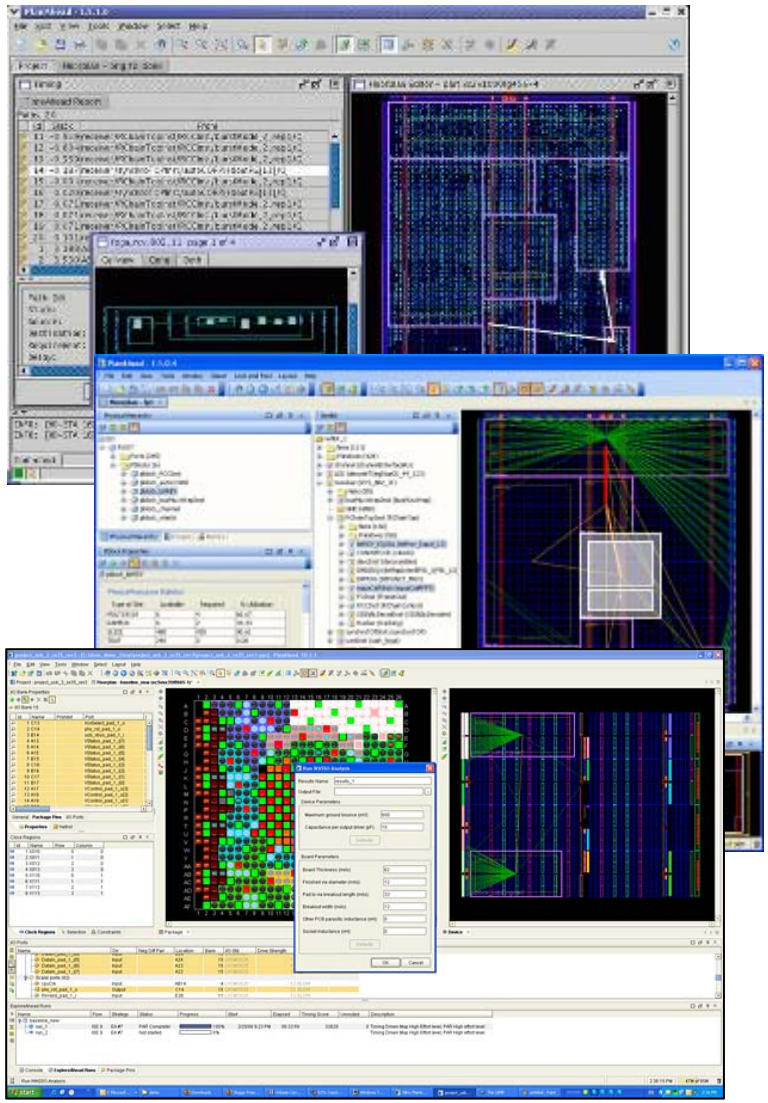
*Smart*Compile Technology

- *Smart*Preview
  - Provides visibility into implementation
  - Create bitstream for lab debug
  - Preserve latest results as snapshot and continue processing
- *Smart*Guide
  - Timing preservation in the midst of changes
  - Average 2x to 4x faster re-implementation runtimes for small design changes
- Partitions
  - Implementation preservation in the midst of changes
  - Allows flexibility to preserve routing, placement, synthesis



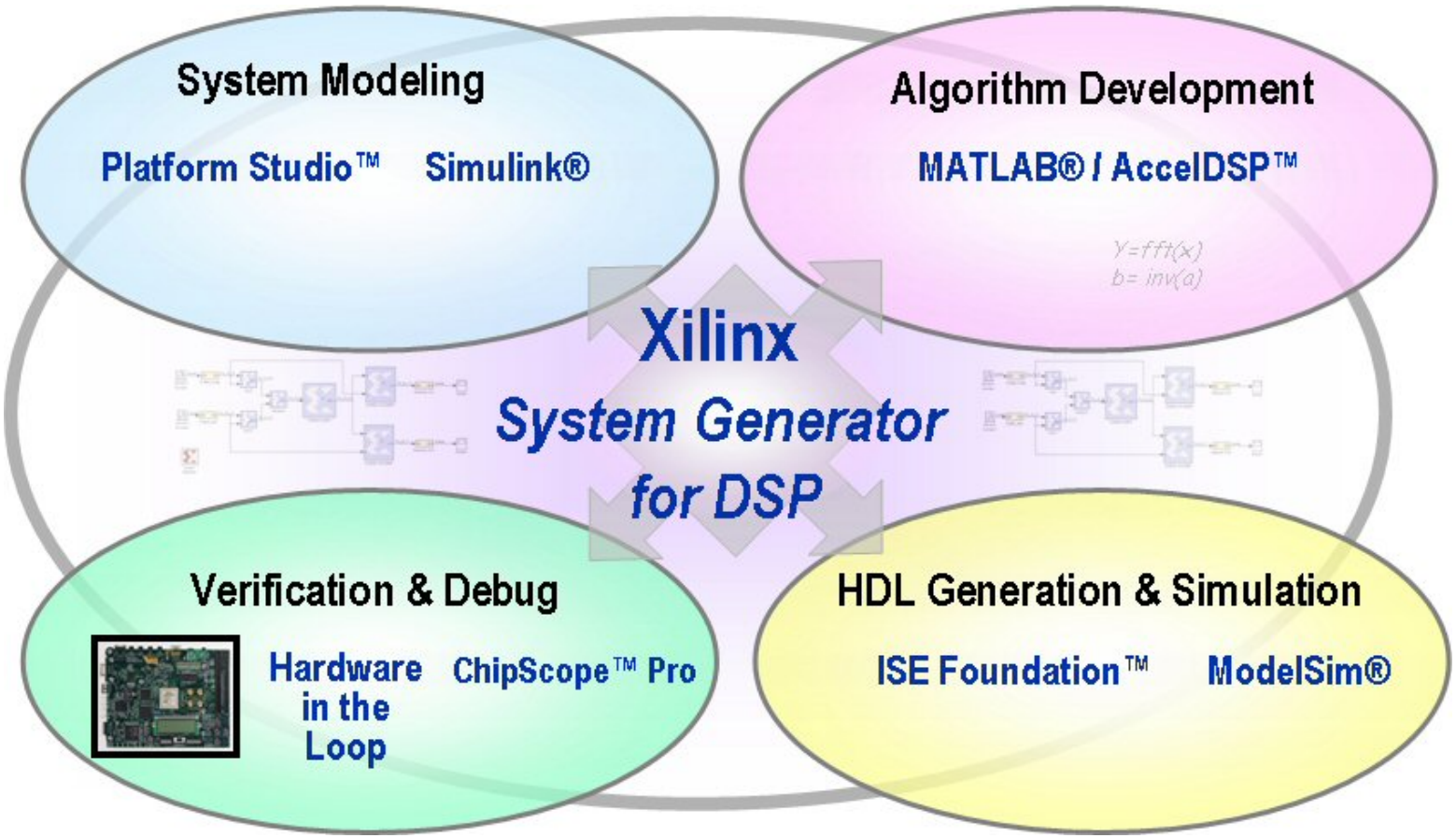
PlanAhead: Floorplanning & More

- Increase performance through hierarchical floorplanning
  - Floorplan prior to physical implementation
  - Guide place and route toward better results
  - Easily view utilization of hierarchy
  - Create Area Constraints quickly
- Analyze Multiple Results from ISE
  - Highlight failing timing paths from post-route timing analysis
- Analyze timing early through TimeAhead
  - Quickly identify, select and constrain critical path logic
- ExploreAhead
  - Run multiple implementations with different implementation switches.
- Simplify managing complex interface between FPGA and PCB with PinAhead
  - Facilitates early and intelligent pinout definition
  - Performs WASSO & Design Rule Checks early in design cycle
  - HDL & CSV Import – Export



www.xilinx.com/planahead

System Generator and AccelDSP



www.xilinx.com/dsp

Intellectual Property Cores

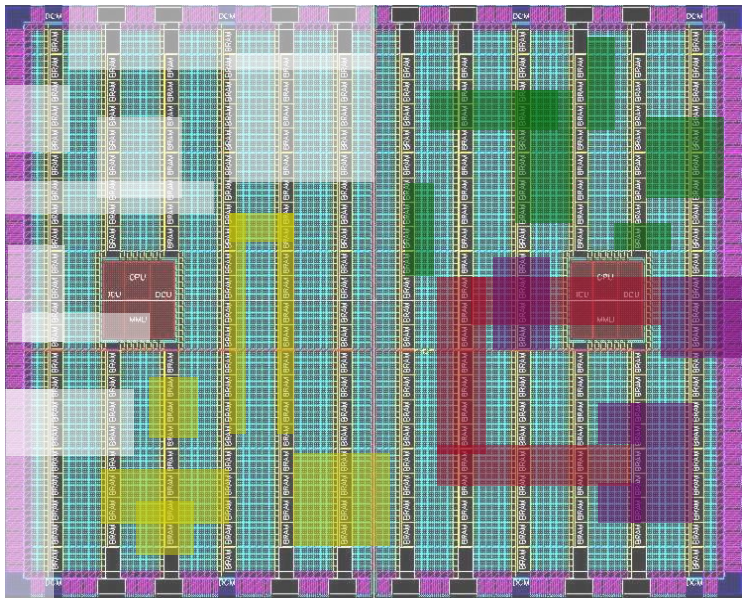
SystemIO

Parallel	Serial
PCI	10 &1 GE MACs
PCI-X	Ethernet PHYs
SPI-4	XAUI
SPI-3	PCI Express
XGMII	Aurora
Many more …	Many more …

General Purpose

CORE Generator
- Building Blocks
- Memory Generators
- IOB Configurations
- Arithmetic and Shifters
- Registers
- Buffers
- Many More …

Optimized for Performance or Area



DSP & Math

Advanced	Math
Reed-Solomon	Multipliers
Turbo Codecs	MAC
Virterbi	Divider
Video	Filters
Wireless	CORDIC
Many More …	Many more …

Processor

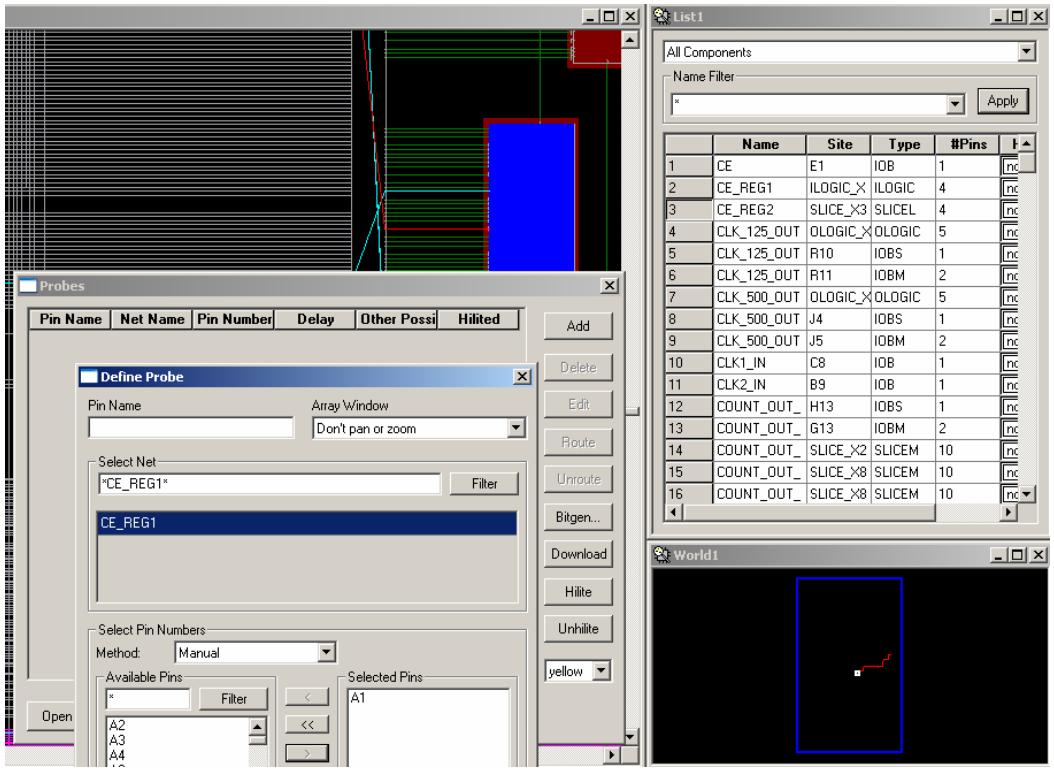
Peripherals	Infrastructure
Interrupt Controller	CoreConnect Bus
UARTs	Arbiter
Timer	Bridge
GPIO	Memory controllers
SPI	Soft processors
Many more …	Software IP
	Many more …

http://www.xilinx.com/ipcenter

http://www.xilinx.com/memory

Signal Probing with FPGA Editor

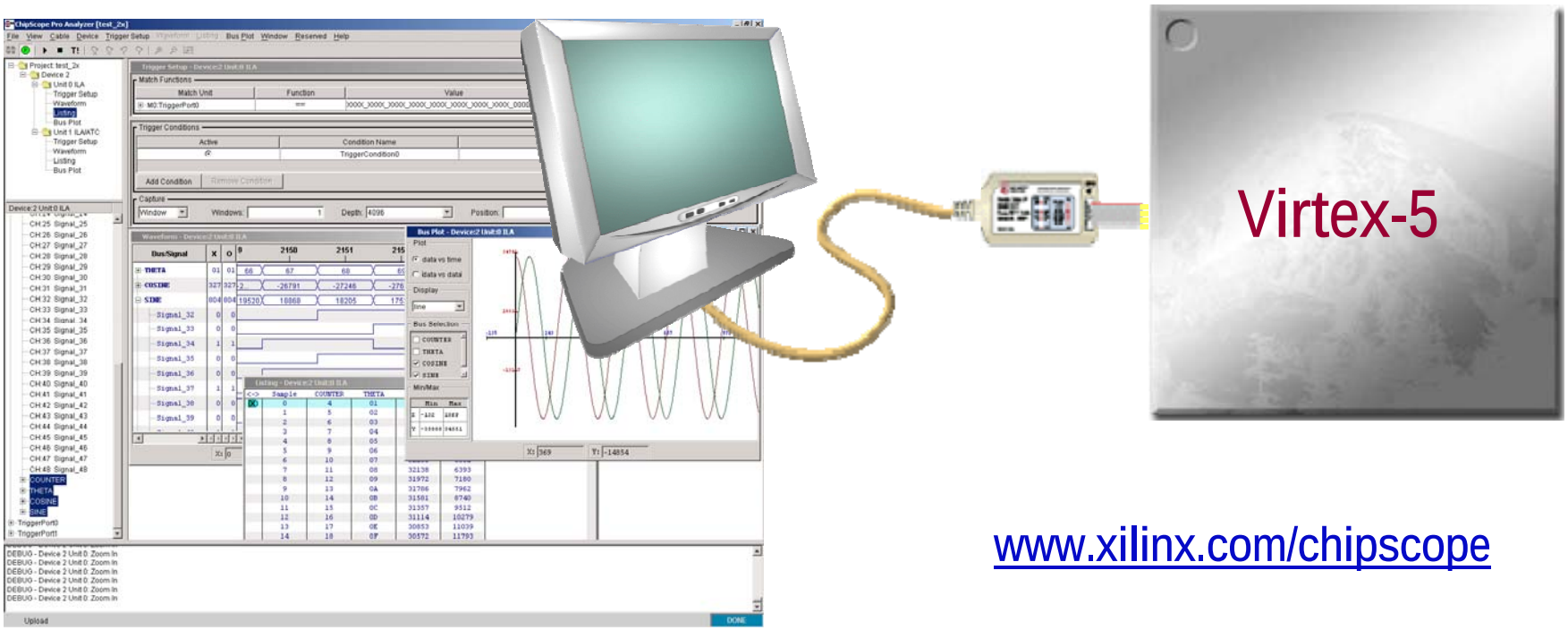
- FPGA Editor shows design layout on device
- Probe internally in design without rerunning implementation
- Find internal signal and route to spare pin
  - Automatically
  - Manually
- Delay from probe point to selected pin automatically reported
- Good for probing a handful of signals




```
Found compatible IO standard "LVTTL" at site "A1".
Building the delay mediator...
  Routed net to A1, pin delay =  1.106ns
Probe of net "CE_REG1" routed to A1 with delay  1.106ns
```

Probe Route Delay

Chipscope Pro Logic Analyzer

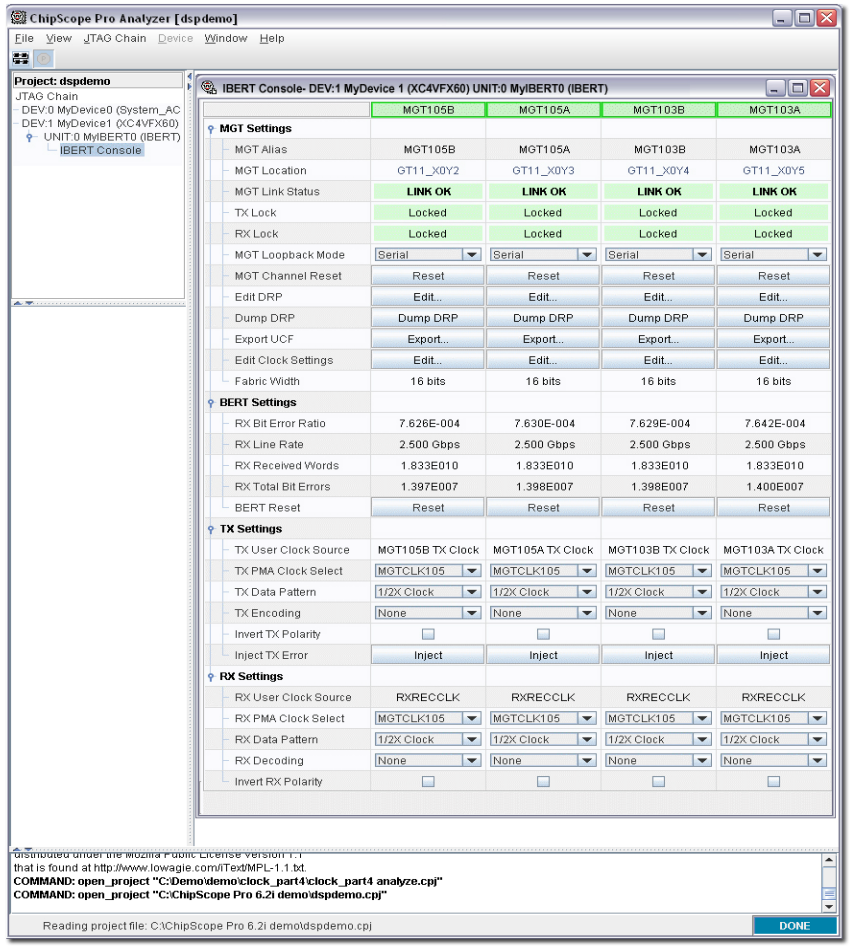


www.xilinx.com/chipscope

- Access ChipScope cores via JTAG or user-defined Trace port
- Configure FPGA, define trigger conditions, and view data
- Chipscope uses unused BRAM in the design to store data
- Not getting enough data? Use Agilent scope with FPGA Dynamic Probe

ChipScope Pro Serial IO Toolkit

- ChipScope Pro IBERT core embedded within the design to provide on-chip access
- Real-time control of each GTP
  - GTP status and control
  - BERT status and control
  - Adjust clock settings and line rate
  - Control TX and RX settings
- Edit MGT attributes or DRP directly
  - Dump DRP attributes to screen
  - Dump DRP attributes to UCF file to include in end design



Summary

- Many different software tools and settings that can help increase the performance and reduce area and power.
- Hard and soft cores that can be leveraged.
- The number one way to increase performance and reduce area and power is to understand the basic features of the target architecture down to the FF and LUT.
- Best way to understand the target architecture is to…… READ, READ, READ and then READ some more!!!

Appendix

Where to find more information

- Xilinx Support Home Page
  - mysupport.xilinx.com
- Virtex-5 FPGA Data Sheet: DC and Switching Characteristics
  - http://www.xilinx.com/support/documentation/data_sheets/ds202.pdf
- Virtex-5 FGPA User Guide
  - http://www.xilinx.com/support/documentation/user_guides/ug190.pdf
- Virtex-5 FPGA XtremeDSP Design Considerations User Guide
  - http://www.xilinx.com/support/documentation/user_guides/ug193.pdf
- Virtex-5 FPGA XtremeDSP Design Considerations User Guide
  - http://www.xilinx.com/support/documentation/user_guides/ug198.pdf

New Video Demos

Streaming videos are available at http://www.xilinx.com/design.

- Improving Design Performance with PlanAhead
- Optimizing Implementation Results using ExploreAhead
- Improving the FGPA on PCB Integration with PinAhead
- Partial Reconfiguration Design using PlanAhead
- Get the Most Out of Your Design Using XST Synthesis Strategies
- Reduce FPGA Verification Time Using New Simulation Features
- Improve Productivity Using Multiple Constraint Files
- Simplify Entry and Analysis of I/O Timing Constraints
- Improve Time-to-Market Using Partitions and SmartGuide
- Improve DSP and Embedded Design Productivity
- Optimize FPGA Performance Using Goals, Strategies, and SmartXplorer
- Improve Productivity Using the EDA Standard Tool Command Language (Tcl)
- Fine-Tune FPGA Power Budgets Using New Power Analysis and Optimization
- Video demo for XPE: http://www.demosondemand.com/clients/xilinx/001/page/index_destools.asp
- Improve Configuration Ease of Use with Project Navigator and iMPACT