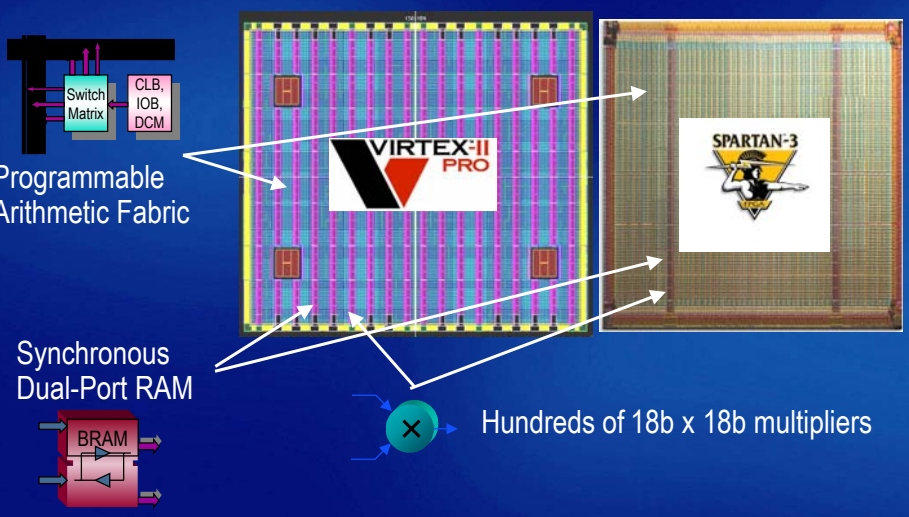
Agenda

- Introduction
  - FPGA DSP platforms
  - Design challenges
  - New programming models for FPGAs
- System Generator
  - Getting your math into an FPGA
  - Co-simulation interfaces
  - Incorporating HDL modules
  - Implementing control circuits
- The future

The DSP Hardware Platforms



Parallelism beats clock speed

What Makes a Platform Successful?

- Device technology is only part of the solution
- Examples of successful platforms:
  - Intel x86 → Microsoft Windows
  - Sun Sparc → Solaris
  - TI TMS320Cxxx → DSP/BIOS, CCS
  - Virtex-II Pro/Spartan-3 → HDL + ISE + …?
- Challenge: Expand the FPGA software platform to support the DSP application developer

Design Challenges

- Lowering the barrier to entry for FPGA-based DSP
  - Simplify traditional FPGA design flows
  - Target the application programmer
  - "Edit, compile, debug" programming models
  - Streamline clerical tasks, expand functionality where possible
- Delivering tools for architecture exploration and design
  - Support quick design iterations
  - Generate efficient hardware from system model
- Pulling hardware into the design loop
  - Provide pushbutton flow for hardware acceleration

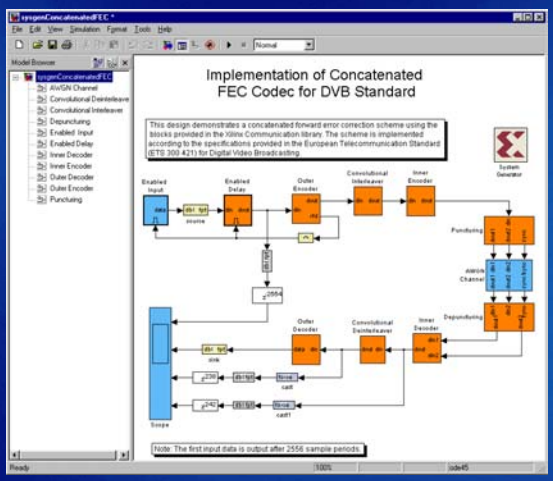

A New Way of FPGA Programming

- A platform-based programming model for FPGAs
  - System level abstractions in Simulink and MATLAB
    - Mathematics, sampled data systems
  - Automatic code generation of *efficient* hardware
- Single "source" for entire development cycle
  - Refine high level abstractions only as needed
    - Minimize verification requirements
  - Allow the user to work in the language of the problem
    - Block diagrams, MATLAB, HDLs, C, assembly, …
- Leverage existing technologies
  - System level software tools and programming languages
  - Physical design tools & IP libraries
  - FPGA hardware platforms


System Generator for DSP

- Library-based, visual data flow
- Polymorphic operators
- Arbitrary precision fixed-point
- Bit and cycle true modeling
- Multi-rate signal processing

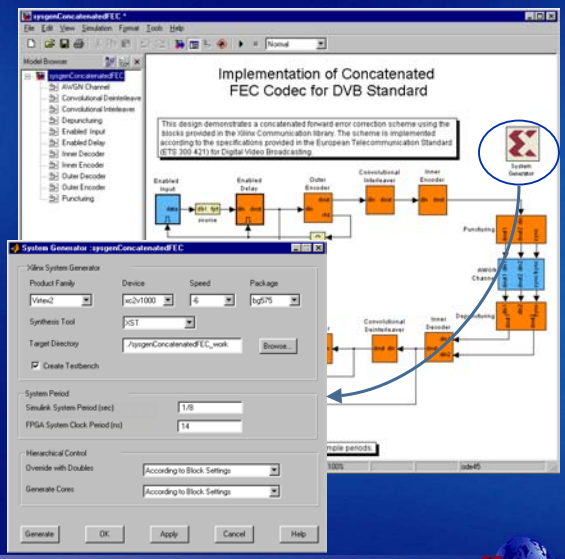
- Seamlessly integrated with Simulink and MATLAB
  - Type and rate propagation
  - Test bench and data analysis
- Automatic code generation
  - Synthesizable VHDL
  - IP cores
  - HDL test bench
  - Project and constraint files

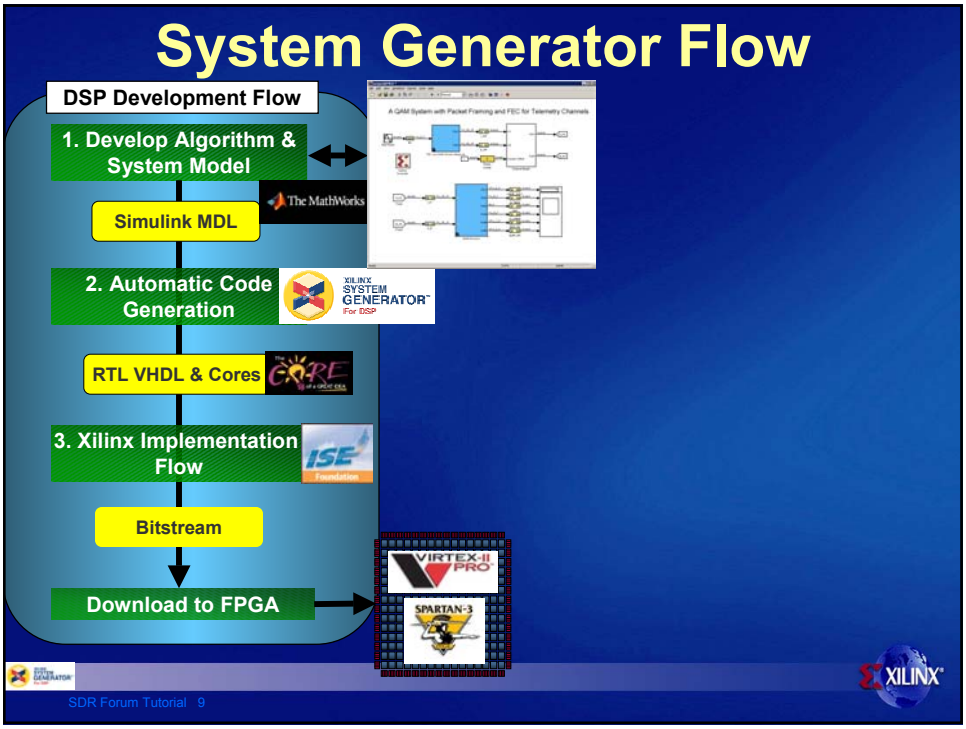


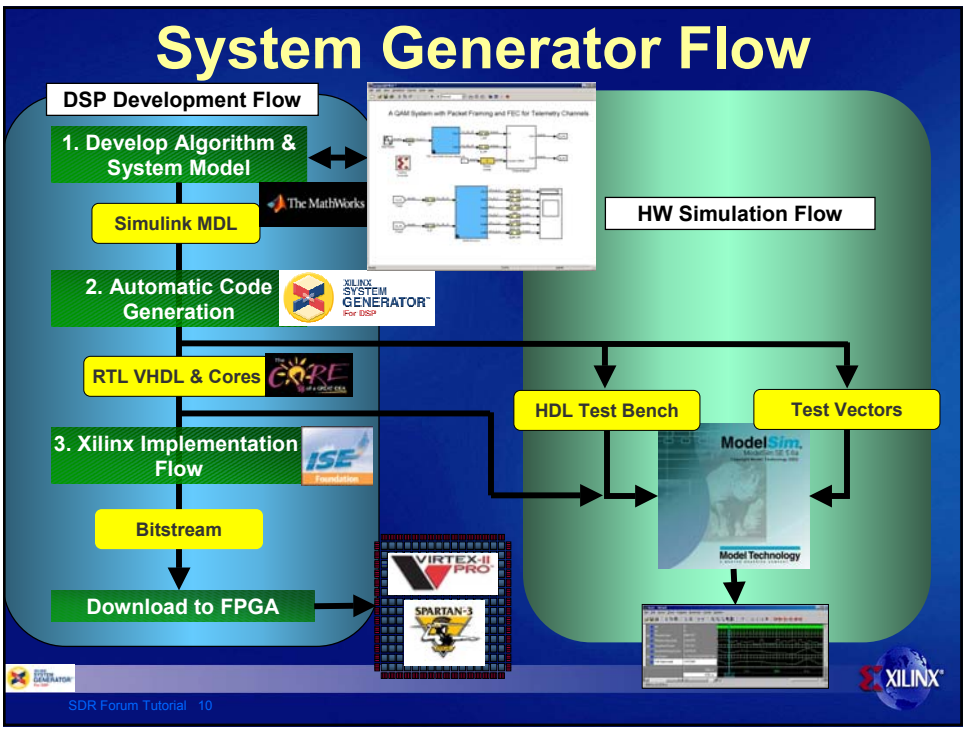
System Generator for DSP

- Library-based, visual data flow
- Polymorphic operators
- Arbitrary precision fixed-point
- Bit and cycle true modeling
- Multi-rate signal processing

- Seamlessly integrated with Simulink and MATLAB
  - Type and rate propagation
  - Test bench and data analysis
- Automatic code generation
  - Synthesizable VHDL
  - IP cores
  - HDL test bench
  - Project and constraint files

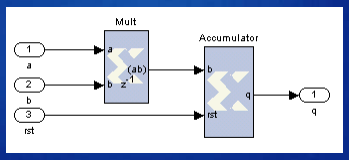






Implementing Math in an FPGA

- A multiply-accumulate engine



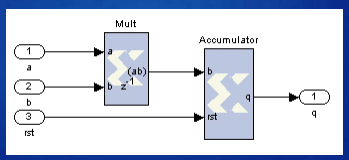
  - Select precision to avoid overflow
  - Minimize resources required

$$|y_n| = \left|\sum_{i=0}^{N-1} h_i x_{n-i}\right| \le \sum_{i=0}^{N-1} |h_i x_{n-i}| \le 2^m \sum_{i=0}^{N-1} |h_i|$$

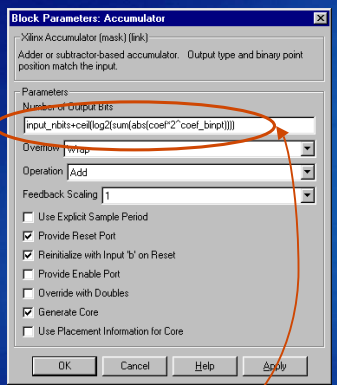
$$Accumulator\ Width = m + \lceil \log_2 \sum |h_i| \rceil$$

Implementing Math in an FPGA

- A multiply-accumulate engine

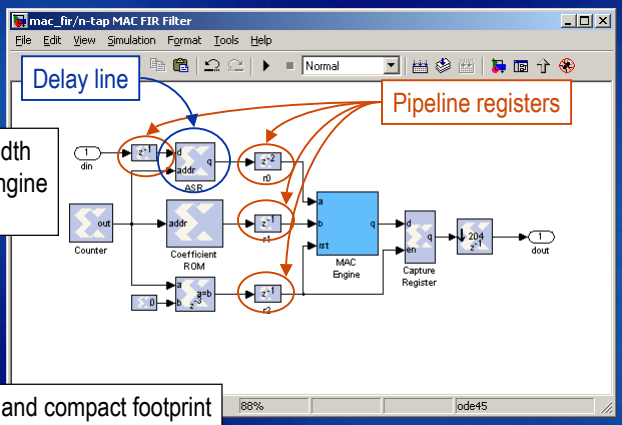


  - Select precision to avoid overflow
  - Minimize resources required
  - Parametric implementation



$$Accumulator\ Width = m + \lceil \log_2 \sum |h_i| \rceil$$

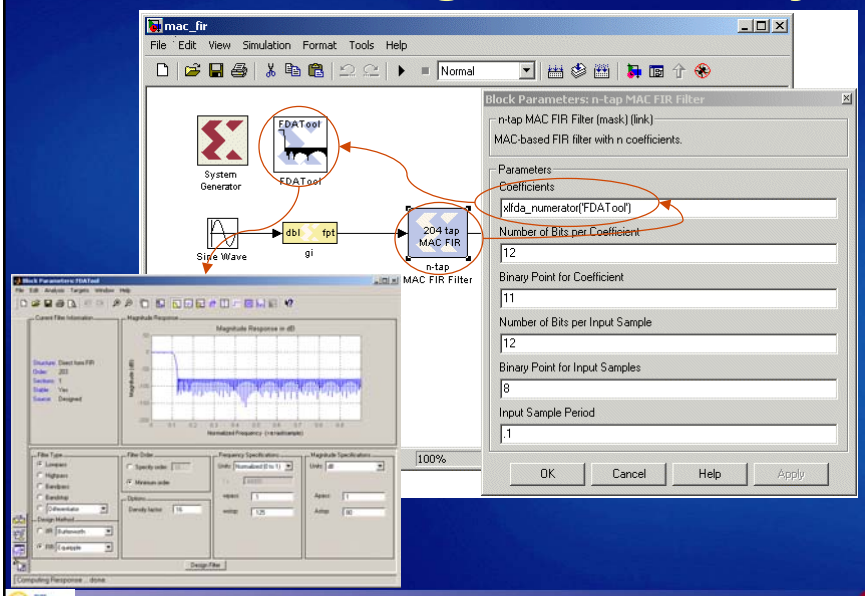
Single-MAC FIR Filter



FPGA memory bandwidth keeps the arithmetic engine fully occupied

High performance and compact footprint
- 85 slices xc2v250-6
- 1 18x18 multipliers, 1 BRAM
- 213 MHz (12 bit data & coefficients)

Filter Design Made Easy



MATLAB Customization

- CORDIC processor
- Versatile family of algorithms for computing functions
  - Arctan, square root, division, logarithm, …
- Shift-and-add architecture
- Well-suited to FPGA implementation

Iteration Equations

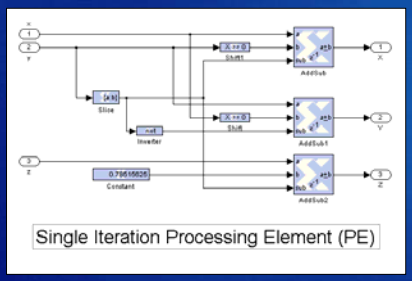
$x_{i+1} = x_i + y_i \delta_i 2^{-i}$

$y_{i+1} = y_i - x_i \delta_i 2^{-i}$

$z_{i+1} = z_i + \delta_i \tan^{-1}(2^{-i})$

$y \rightarrow 0$

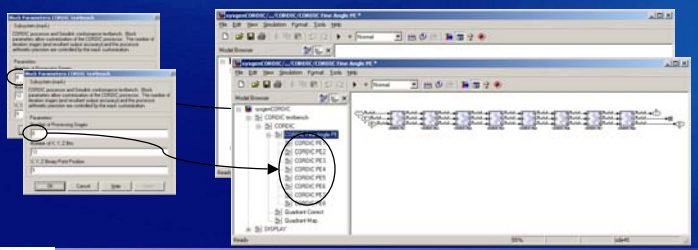
$\delta_i \varepsilon \{-1, 1\}$



Single Iteration Processing Element (PE)

MATLAB Customization (II)

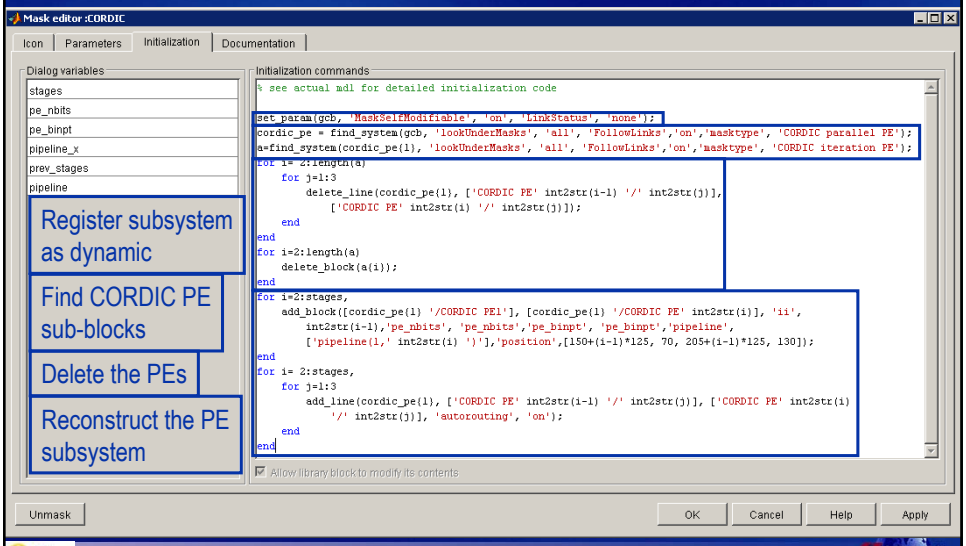
- CORDIC accuracy depends on the number of PEs
  - Trade area for precision
- Use MATLAB to parameterize the processor
  - Propagate parameters down to blocks
    - Data types, constants
  - Conditionally and iteratively instance PEs



4 PEs, 12-bit data
193 LUTs, 213 MHz

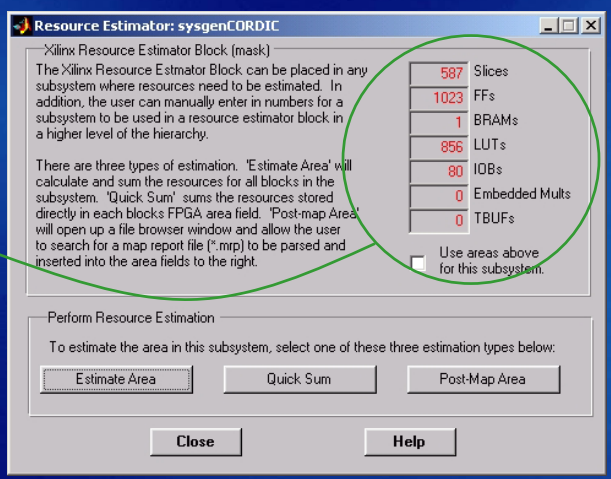
8 PEs
325 LUTs, 211 MHz

Programmatic Diagrams



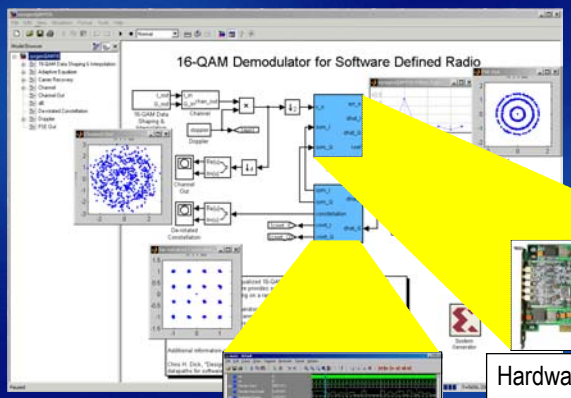
System Resource Estimation

- Estimate FPGA resources directly from Simulink
  - Every block
  - Any level in the system hierarchy
  - Invoke mapper for more accuracy



Co-Simulation Interfaces

Simulink Extensions for HW



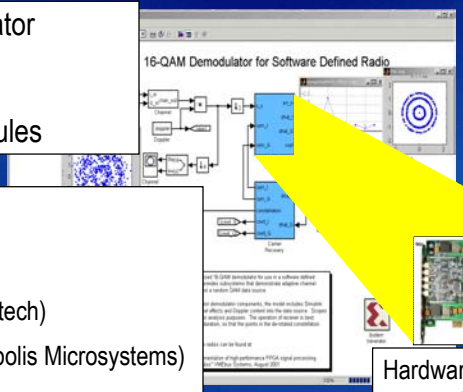
•HDL Configuration Wizard
•MATLAB compilation block

HDL co-simulation

Hardware in the loop co-simulation

HW Co-Simulation

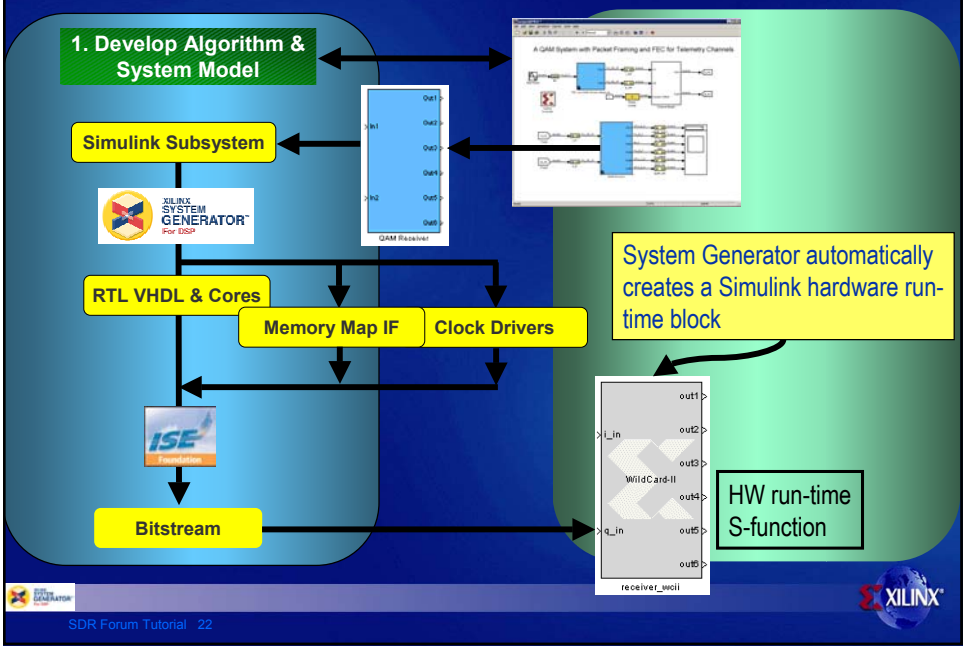
- Simulink hardware accelerator
- Verify design in hardware
- Simulate HDL & EDIF modules



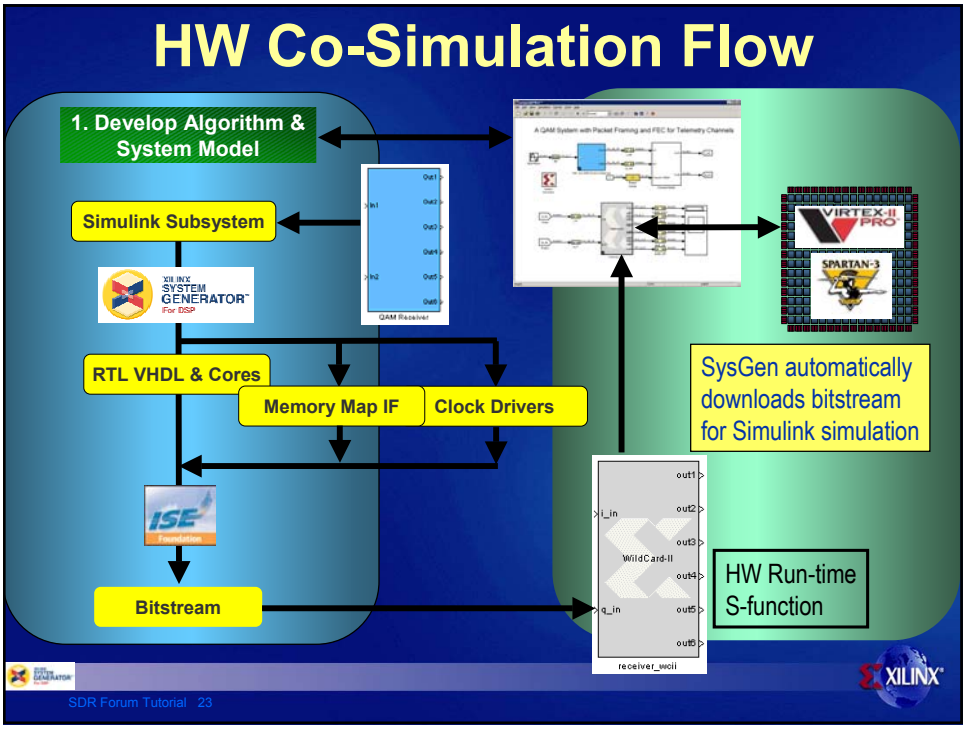
- Simulink push-button flow
- Supported platforms
  - –Xilinx XtremeDSP kit (Nallatech)
  - –Wildcard/Wildcard-II (Annapolis Microsystems)
  - –XRC (Alpha Data)
  - –Signal Master (Lyr Signal Processing)

Hardware in the loop co-simulation

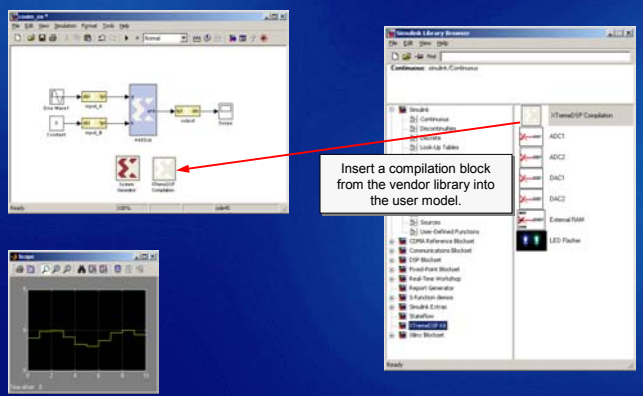
HW Co-Simulation Flow



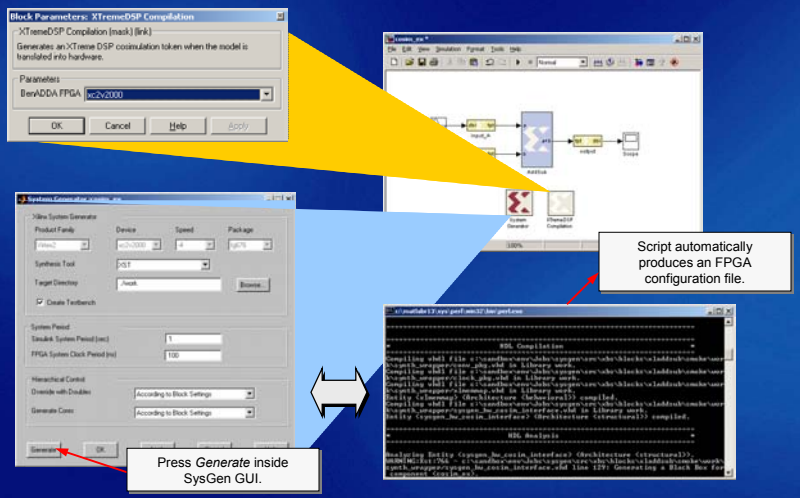
HW Co-Simulation Flow



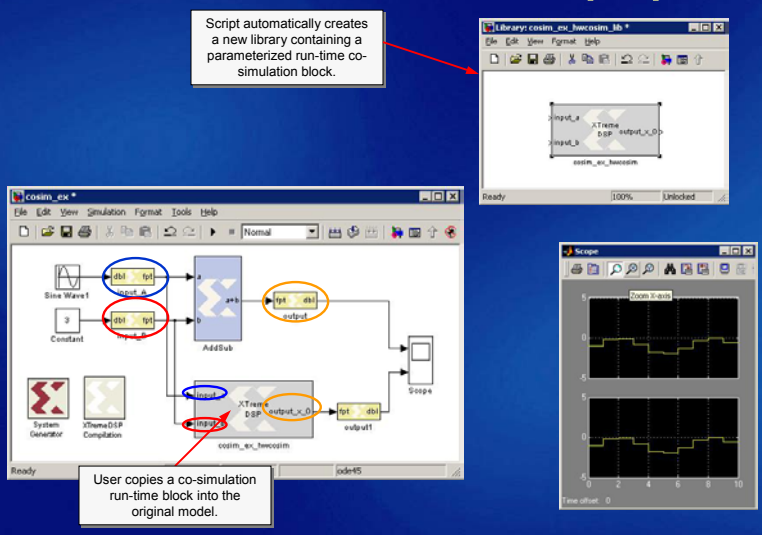
HW Co-Simulation (II)



HW Co-Simulation (III)



HW Co-Simulation (IV)

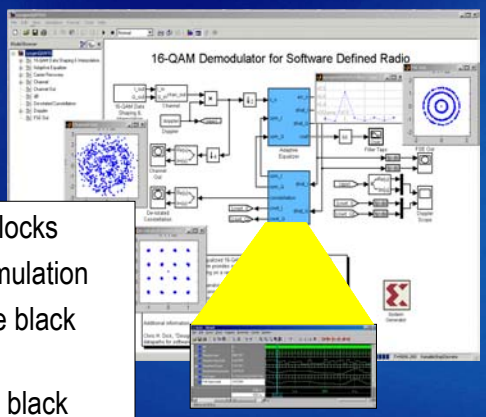


Incorporating HDL Modules in System Generator

Why HDL?

- System Generator does not replace HDL design
  - Legacy HDL modules
  - Control circuitry still sometimes easier in HDLs than SysGen
  - Sometimes require access to low-level FPGA resources (e.g. MUXF6, XORCY, multi-phased clocking)
  - Bit and cycle accurate modeling may be undesirable
    - PCI interface, SDRAM interface, etc.
- It can still be desirable to deliver HDL interface modules during code generation
- In the early days of compilers, *real*™ programmers wrote assembly code
  - System Generator designs can be extremely efficient

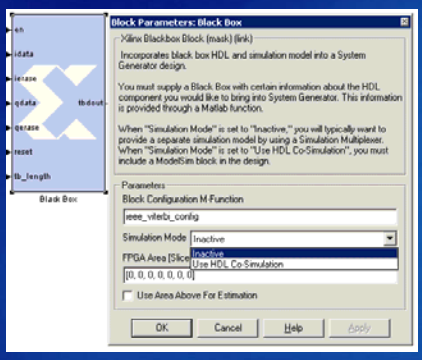
HDL Co-Simulation



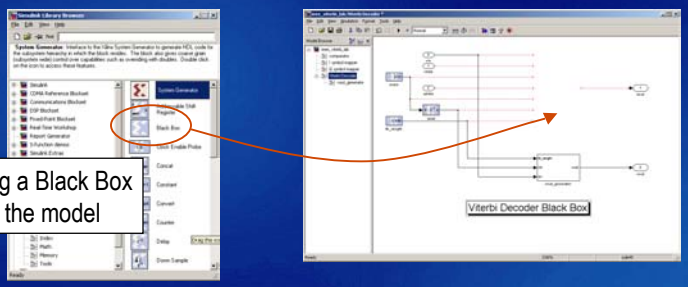
- Import legacy HDL and user IP blocks
- Simulink testbenches for HDL simulation
- Configuration Wizard to configure black box HDL
- Single HDL simulator for multiple black boxes

Black Box Interface

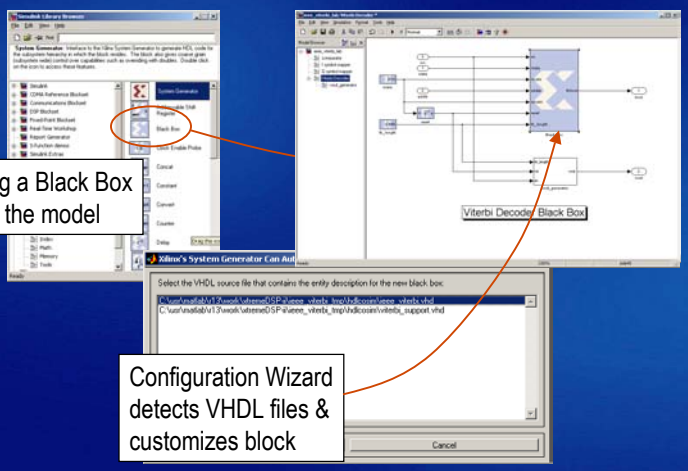
- Block for representing non-SysGen functions
  - Typically HDL-based
  - M-code configuration function
- Simulation options
  - Simulink block or subsystem
  - S-function provided by user
  - HDL co-simulation
- Implementation files
  - HDL, EDIF, constraint files,…
- HDL co-simulation is closely related to Black Box functionality



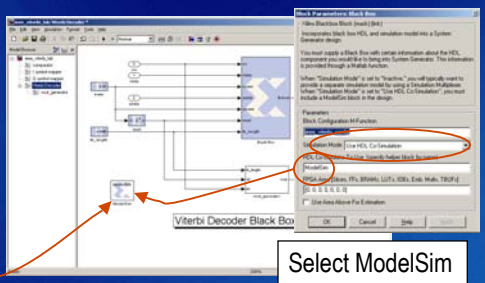
HDL Co-Simulation (II)



HDL Co-Simulation (II)



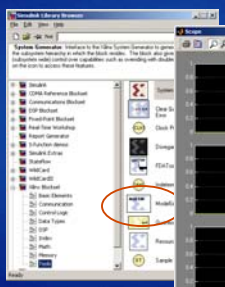
HDL Co-Simulation (III)

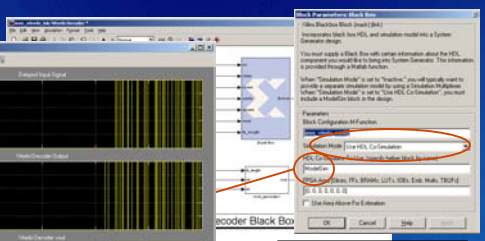


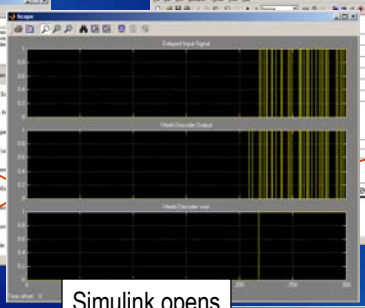
Drag a ModelSim block into the model

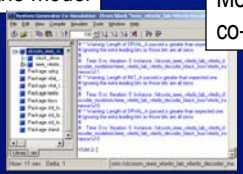
Select ModelSim Simulation Mode

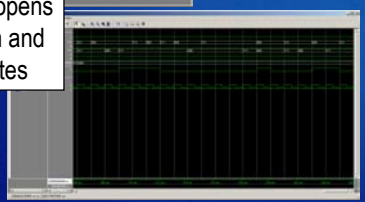
HDL Co-Simulation (III)











Drag a ModelSim block into the model

Simulink opens ModelSim and co-simulates

Select ModelSim Simulation Mode

HW Co-simulation

- HDL simulator provides access to internal state of an HDL module
- If full visibility is not required, can use HW co-simulation interface to implement the Black Box
- Hardware accelerator for Simulink
- Use ChipScope internal logic analyzer for real-time debugging on free-running clock

Implementing Control Circuits in System Generator

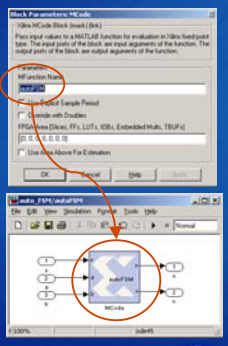
Finite State Machines

- Common form of control circuitry
- Many options in System Generator
  - Build out of the Xilinx Blockset primitives
  - Import HDL into Simulink via co-simulation
  - MATLAB code using the System Generator m-code block
  - Embedded microcontrollers

MATLAB M-Code Compilation

- System Generator m-code block supports MATLAB control flow constructs
  - Nested branches (if, then, else, switch)
  - Variable assignment
  - Logical, comparison, some arithmetic operators
- Well-suited for FSM transition functions
- Code generator compiles into equivalent RTL VHDL

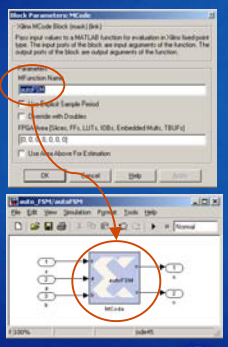
```
function [n, v] = autoFSM(s, a, b)
  stopped = 0;  moving = 1;
  switch s
   case stopped
    if a & ~b     n = 1, v = 20;
    else          n = 0, v = 0;
    end
   case moving
    if a | ~b     n = 1, v = 20;
    else          n = 0, v = 0;
    end
   otherwise
    n = 0, v = 0;
  end
```



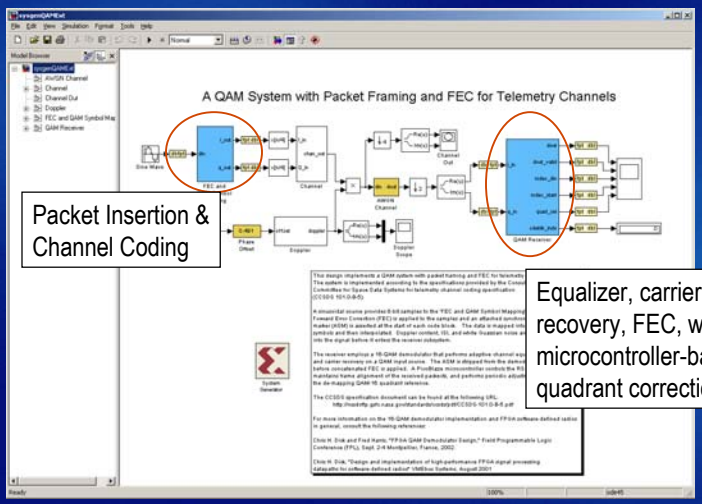
MATLAB M-Code Compilation

- System Generator m-code block supports control flow constructs in MATLAB
  - Nested branches (if, then, else, switch)
  - Variable assignment
  - Logical & comparator operators
- Well-suited for FSM transition functions
- Code generator compiles into equivalent RTL VHDL

```
architecture behavior of auto_fsm_xlmcode_x_0 is
begin
  process_x_0 : process (s, a, b) is
    begin
      s_x_0 := std_logic_vector_to_unsigned (s);
      a_x_0 := a = "1";
      b_x_0 := b = "1";
      case_var := s_x_0;
      case case_var is
        when "0" =>
          if a_x_0 and (not b_x_0) then
            v_x_2 := std_logic_vector_to_unsigned ("10100");
            n_x_2 := std_logic_vector_to_unsigned ("1");
          else
            v_x_2 := std_logic_vector_to_unsigned ("00000");
            n_x_2 := std_logic_vector_to_unsigned ("0");
          end if;
        ...
```

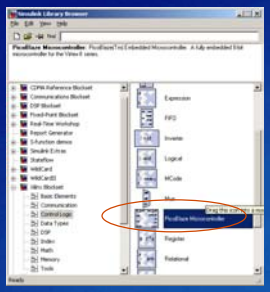


Embedded Microcontrollers



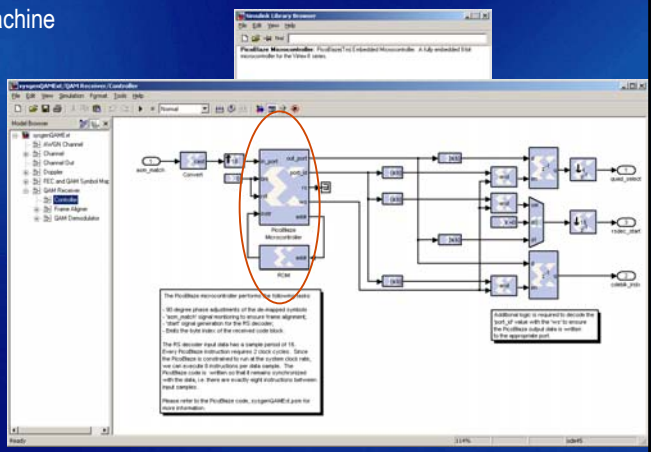
PicoBlaze Controller

- 8-bit programmable state machine
  - Very compact (85 slices)
  - 49 instructions
  - Assembly language
  - 2 clocks/instruction



PicoBlaze Controller

- 8-bit programmable state machine
  - Very compact (85 slices)
  - 49 instructions
  - Assembly language
  - 2 clocks/instruction



- In the QAM demo design
  - Monitors ASM match for frame alignment
  - Quadrant rotation
  - RS-Decoder flow control
  - Emits RS byte index

Other Approaches to FPGA-based DSP

The Visible Spectrum

- Algorithm focus
  - MATLAB-to-gates
    - Accelchip MATLAB compiler
  - C-to-gates
    - Handel-C (Celoxica)
    - Xilinx Forge compiler
- System level focus
  - System-C
    - Synopsys, Co-Ware, Cadence, …
    - Primarily modeling and verification, not implementation
- Real life, i.e. ad hoc flows
  - Traditional HDL-based FPGA design still going strong

Where Is Xilinx DSP Going?

- Additional dedicated hardware functions
  - Beyond PPC 405, 18x18 multipliers, block memories, MGTs
- Increasingly sophisticated intellectual property (hard and soft) as well as reference designs for signal processing
- Embedded processors and software as well as hardware
  - Supervisory functions and protocols
- Design methodologies will continue to support the platform
  - Component-based modeling and deployment
  - Work in the language of the problem


Thank You