



New Directions in Programming FPGAs for DSP

Dr. Jim Hwang
Xilinx, Inc.

Agenda

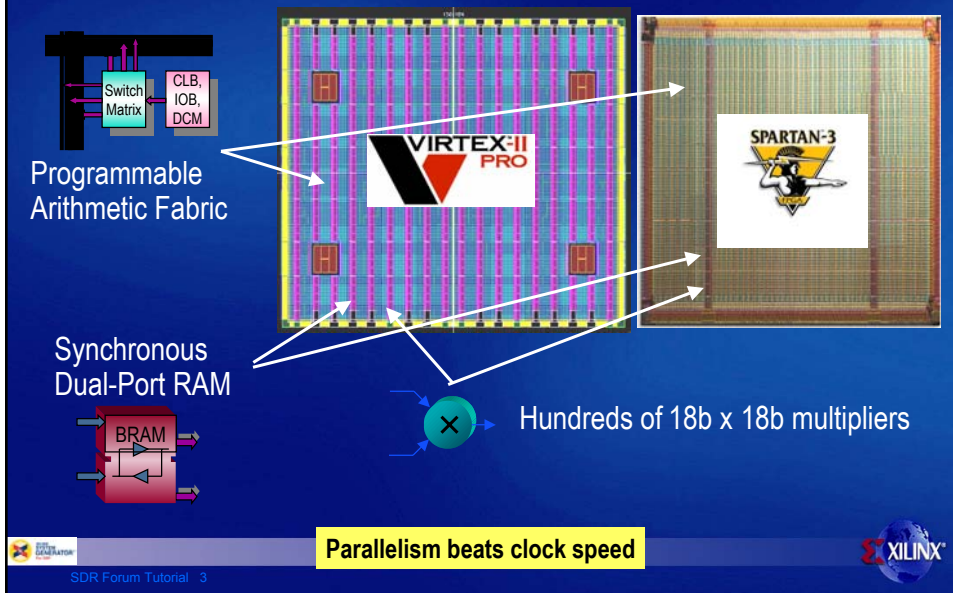
- Introduction
 - FPGA DSP platforms
 - Design challenges
 - New programming models for FPGAs
- System Generator
 - Getting your math into an FPGA
 - Co-simulation interfaces
 - Incorporating HDL modules
 - Implementing control circuits
- The future



SDR Forum Tutorial 2



The DSP Hardware Platforms



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



SDR Forum Tutorial 5



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

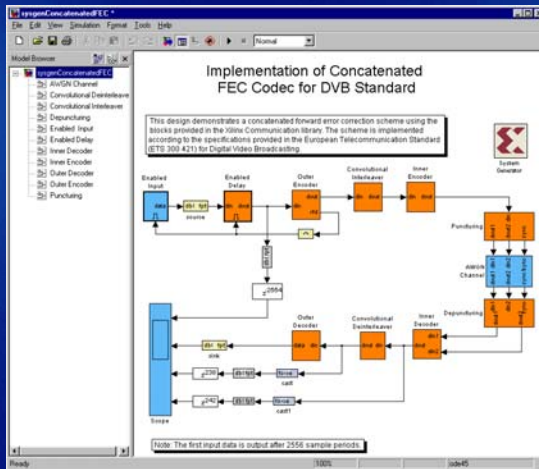


SDR Forum Tutorial 6



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

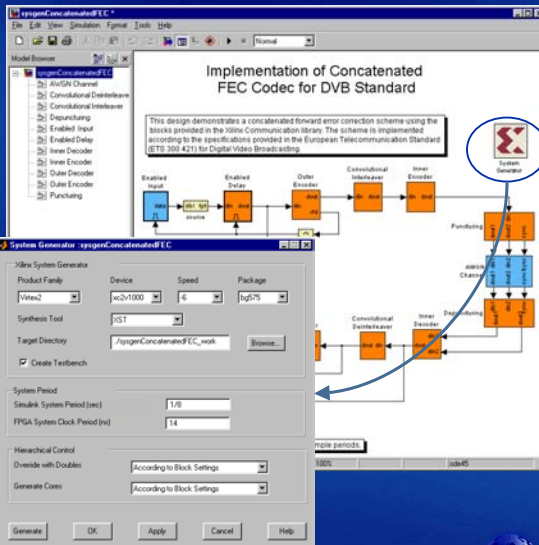


SDR Forum Tutorial 7



System Generator for DSP

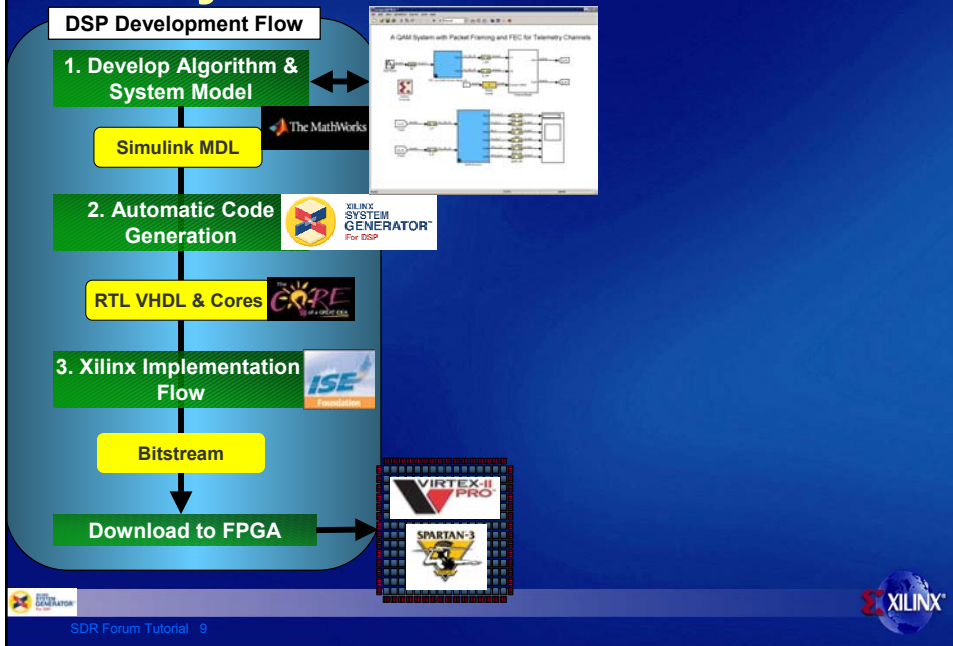
- Library-based, visual data flow
- Polymorphic operators
- Arbitrary precision fixed-point
- Bit and cycle true modeling
- Multi-rate signal processing
- Seamlessly integrated with Simulink and MATLAB
 - Type and rate propagation
 - Test bench and data analysis
- Automatic code generation
 - Synthesizable VHDL
 - IP cores
 - HDL test bench
 - Project and constraint files



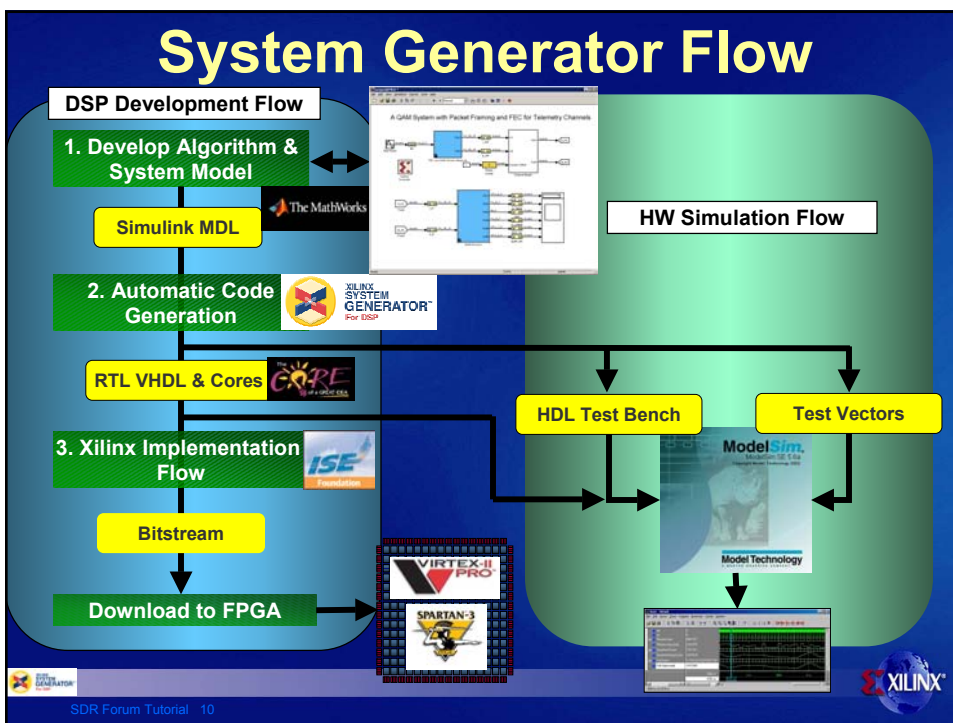
SDR Forum Tutorial 8



System Generator Flow

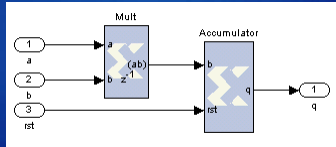


System Generator Flow



Implementing Math in an FPGA

- A multiply-accumulate engine



- Select precision to avoid overflow
- Minimize resources required

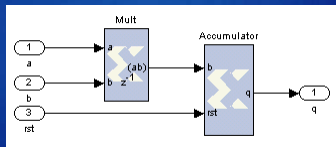
$$\begin{aligned}
 |y_n| &= \left| \sum_{i=0}^{N-1} h_i x_n - i \right| \\
 &\leq \sum_{i=0}^{N-1} |h_i x_n - i| \\
 &\leq 2^m \sum_{i=0}^{N-1} |h_i|
 \end{aligned}$$

$$\text{Accumulator Width} = m + \left\lceil \log_2 \sum |h_i| \right\rceil$$

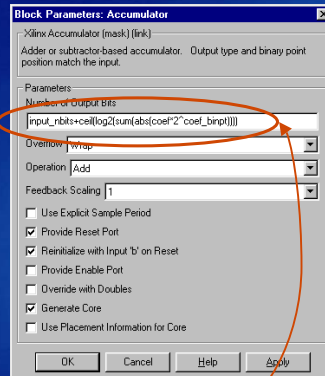


Implementing Math in an FPGA

- A multiply-accumulate engine



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



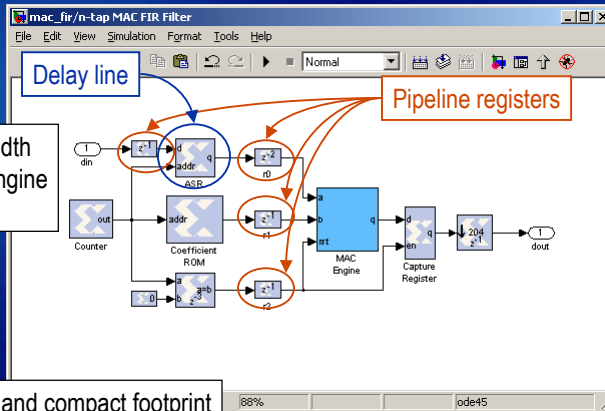
$$\text{Accumulator Width} = m + \left\lceil \log_2 \sum |h_i| \right\rceil$$



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

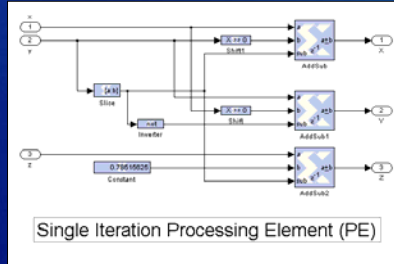
$$x_{i+1} = x_i + y_i \delta_i 2^{-i}$$

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

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

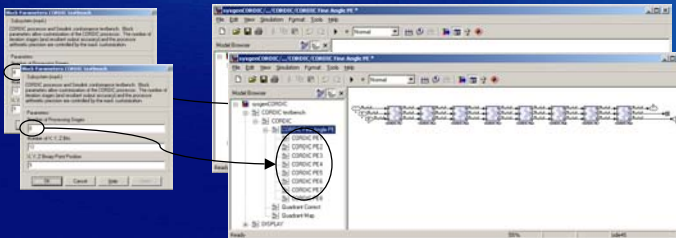
$$y \rightarrow 0$$

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



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

Dialog variables

- stages
- pe_nbits
- pe_binpt
- pipeline_x
- prev_stages
- pipeline

Initialization commands

```
% see actual mdl for detailed initialization code
set_param(gcb, 'MaskScalModifiable', 'on', 'LinkStatus', 'none');
cordic_pe = find_system(gcb, 'lookUnderMasks', 'all', 'FollowLinks', 'on', 'masktype', 'CORDIC parallel PE');
a=find_system(cordic_pe(1), 'lookUnderMasks', 'all', 'FollowLinks', 'on', 'masktype', 'CORDIC iteration PE');
for i= 2:length(a)
    for j=1:3
        delete_line(cordic_pe(1), ['CORDIC PE' int2str(i-1) '/' int2str(j)],
            ['CORDIC PE' int2str(i) '/' int2str(j)]);
    end
end
for i=2:length(a)
    delete_block(a(i));
end
for i=2:stages,
    add_block([cordic_pe(1) '/'CORDIC PE1'], [cordic_pe(1) '/'CORDIC PE' int2str(i)], 'ii',
        int2str(i-1), 'pe_nbits', 'pe_nbits', 'pe_binpt', 'pe_binpt', 'pipeline',
        ['pipeline(1,' int2str(i) ')'], 'position', [150+(i-1)*125, 70, 205+(i-1)*125, 130]);
end
for i= 2:stages,
    for j=1:3
        add_line(cordic_pe(1), ['CORDIC PE' int2str(i-1) '/' int2str(j)], ['CORDIC PE' int2str(i)
            '/' int2str(j)], 'autorouting', 'on');
    end
end
```

Register subsystem as dynamic

Find CORDIC PE sub-blocks

Delete the PEs

Reconstruct the PE subsystem

Unmask OK Cancel Help Apply

SDR Forum Tutorial 17

System Resource Estimation

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

Resource Estimator: sysgenCORDIC

Xilinx Resource Estimator Block (mask)

The Xilinx Resource Estimator Block can be placed in any subsystem where resources need to be estimated. In addition, the user can manually enter in numbers for a subsystem to be used in a resource estimator block in a higher level of the hierarchy.

There are three types of estimation. 'Estimate Area' will calculate and sum the resources for all blocks in the subsystem. 'Quick Sum' sums the resources stored directly in each block's FPGA area field. 'Post-map Area' will open up a file browser window and allow the user to search for a map report file (*.mip) to be parsed and inserted into the area fields to the right.

587 Slices

1023 FFs

1 BRAMs

856 LUTs

80 IOBs

0 Embedded Mults

0 TBUFs

Use areas above for this subsystem.

Perform Resource Estimation

To estimate the area in this subsystem, select one of these three estimation types below:

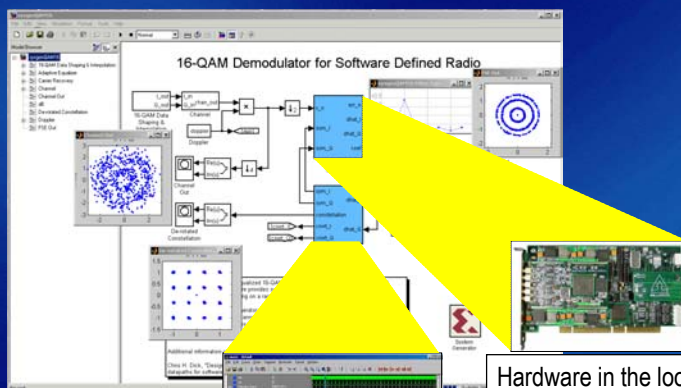
Estimate Area Quick Sum Post-Map Area

Close Help

Co-Simulation Interfaces



Simulink Extensions for HW



- HDL Configuration Wizard
- MATLAB compilation block

HDL co-simulation

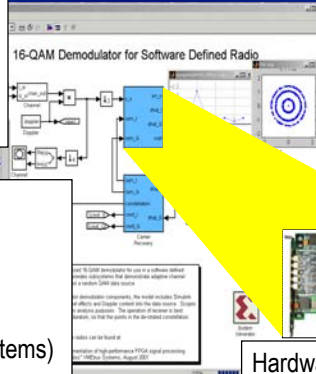
Hardware in the loop
co-simulation



HW Co-Simulation

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

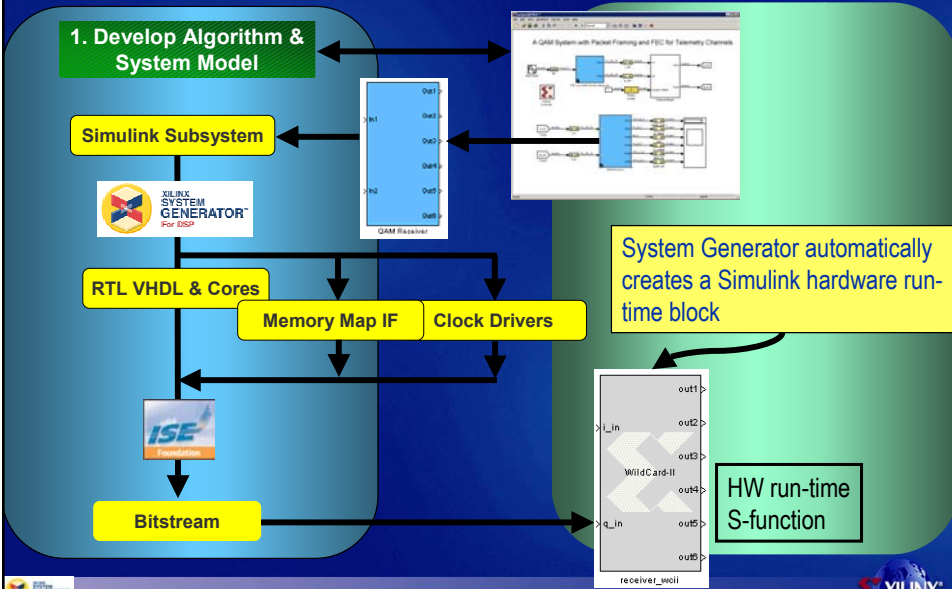
- Simulink push-button flow
- Supported platforms
 - Xilinx XtremeDSP kit (Nallatech)
 - Wildcard/Wildcard-II (Annapolis Microsystems)
 - XRC (Alpha Data)
 - Signal Master (Lyr Signal Processing)



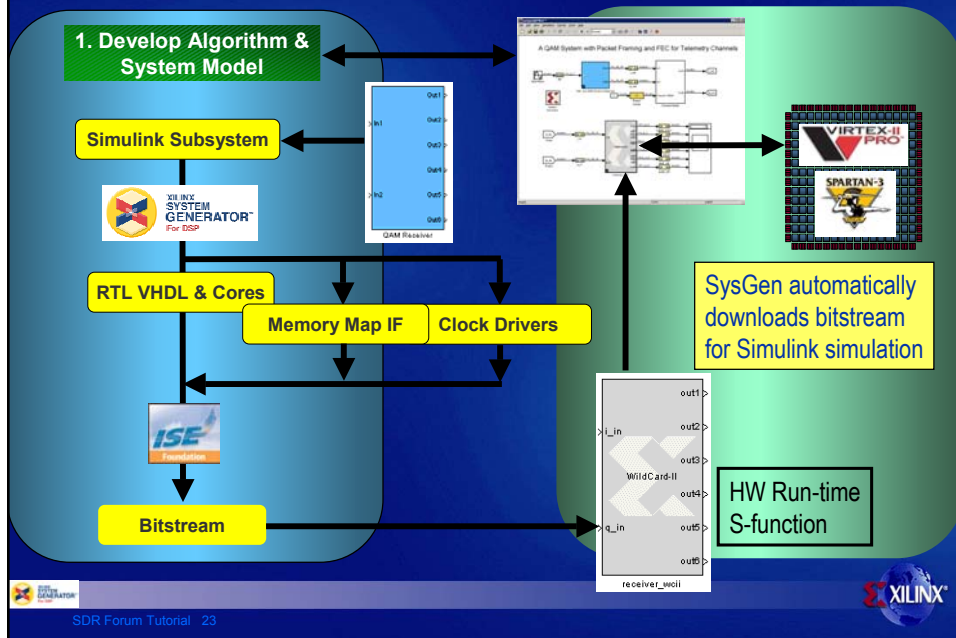
Hardware in the loop
co-simulation



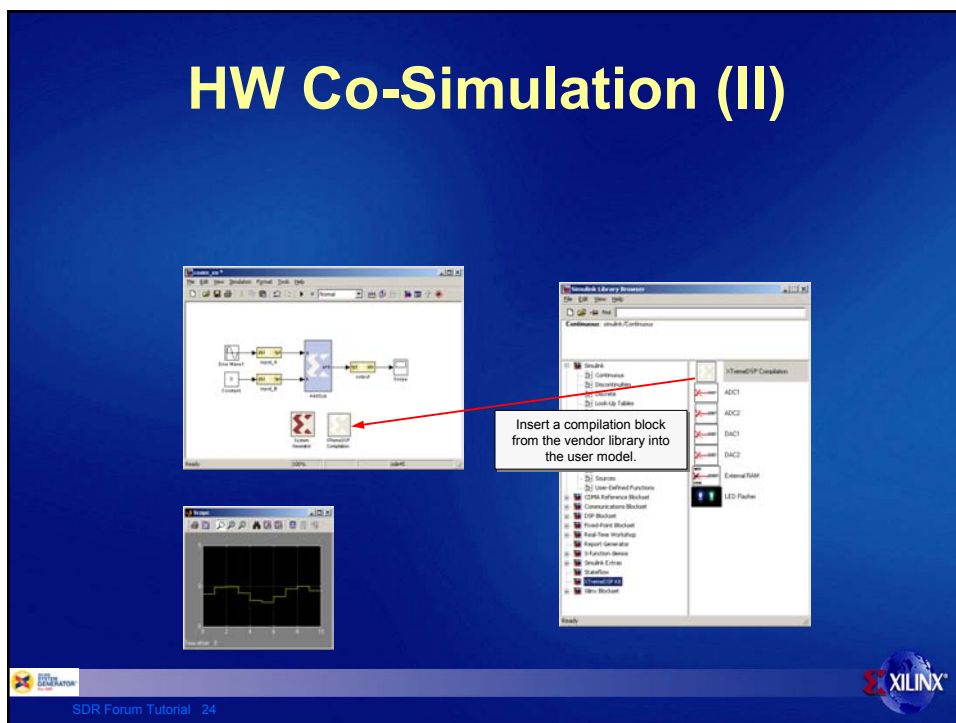
HW Co-Simulation Flow



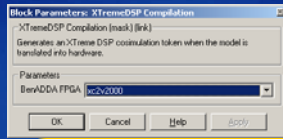
HW Co-Simulation Flow



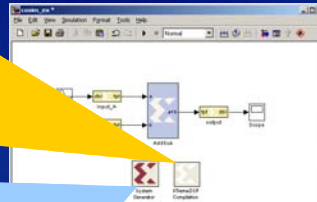
HW Co-Simulation (II)



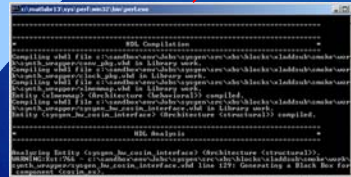
HW Co-Simulation (III)



Press Generate inside SysGen GUI.

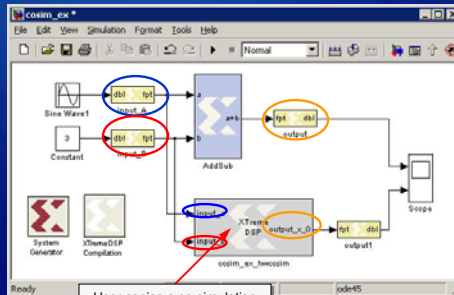
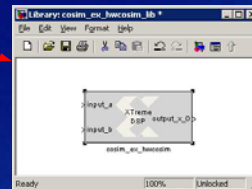


Script automatically produces an FPGA configuration file.

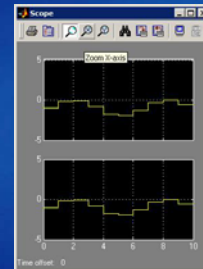


HW Co-Simulation (IV)

Script automatically creates a new library containing a parameterized run-time co-simulation block.



User copies a co-simulation run-time block into the original model.



Incorporating HDL Modules in System Generator



SDR Forum Tutorial 27



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*TM programmers wrote assembly code
 - System Generator designs can be extremely efficient

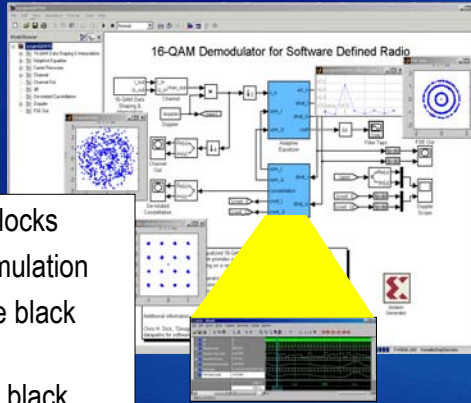


SDR Forum Tutorial 28



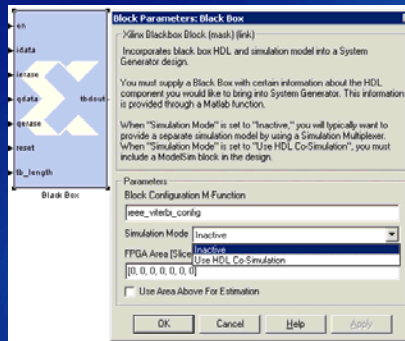
HDL Co-Simulation

- 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

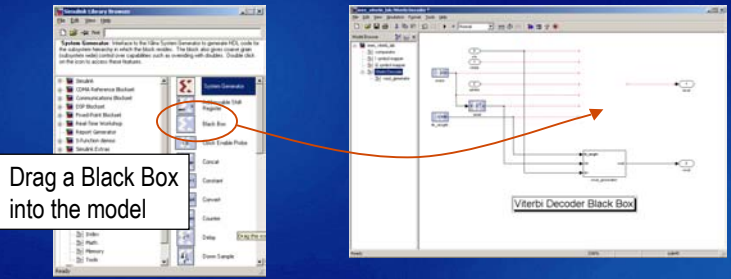


Black Box Interface

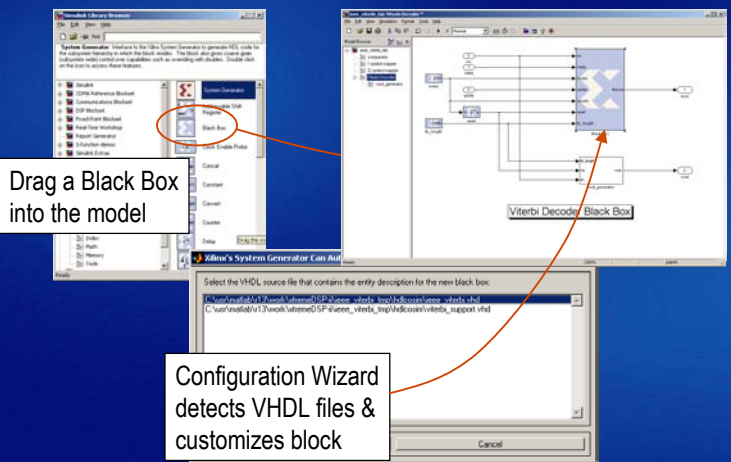
- Block for representing non-SysGen functions
 - Typically HDL-based
 - M-code configuration function
- Simulation options
 - Simulink block or subsystem
 - S-function provided by user
 - HDL co-simulation
- Implementation files
 - HDL, EDIF, constraint files,...
- HDL co-simulation is closely related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



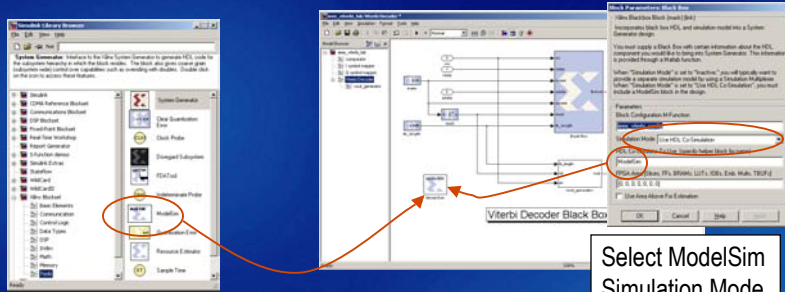
HDL Co-Simulation (II)



HDL Co-Simulation (II)



HDL Co-Simulation (III)

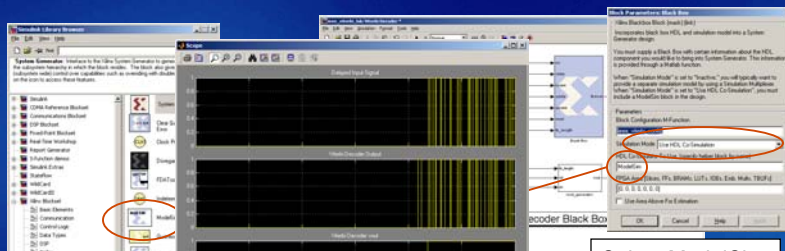


Drag a ModelSim block into the model

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



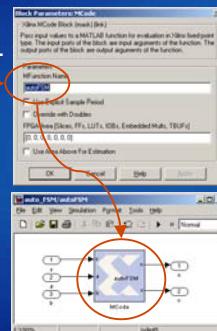
SDR Forum Tutorial 37



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
```



SDR Forum Tutorial 38

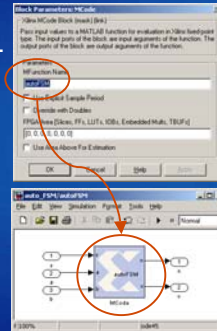


MATLAB M-Code Compilation

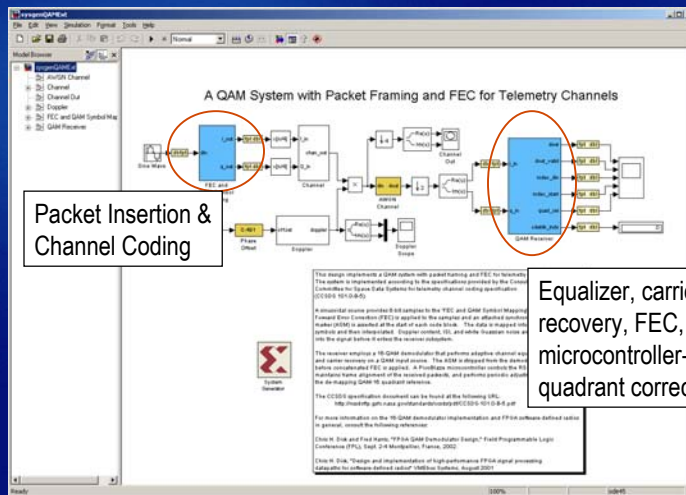
- System Generator m-code block supports control flow constructs in MATLAB
 - Nested branches (if, then, else, switch)
 - Variable assignment
 - Logical & comparator operators
- Well-suited for FSM transition functions
- Code generator compiles into equivalent RTL VHDL

```

architecture behavior of auto_fsm_xlcode_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;
      ...
    end case;
  end process;
end architecture;
    
```

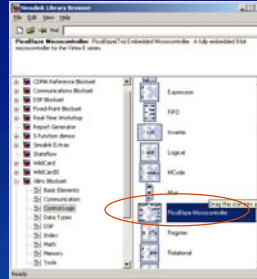


Embedded Microcontrollers



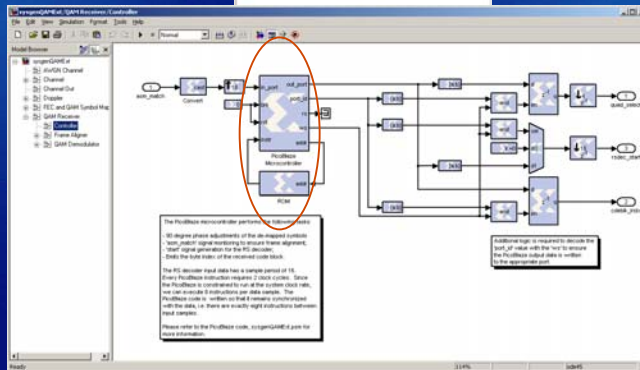
PicoBlaze Controller

- 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



SDR Forum Tutorial 43



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



SDR Forum Tutorial 44



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

