The MathWorks MATLAB&SIMULINK®

Implementation of Hardware and Software Systems using Model-Based Design

Part of DATE'08 Tutorial:
Automatically Realizing Embedded Systems from High-Level Functional Models

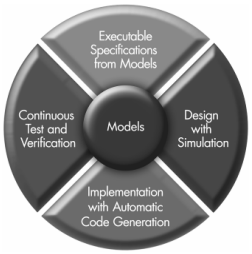
Monday, 10 Mar 2008

Don Orofino, PhD
Director, Signal Processing Development
The MathWorks, Inc.

DATE 08

The MathWorks MATLAB&SIMULINK®

Tutorial Segment



- Discuss use of Model-Based Design (MBD) to implement hardware and software systems
- Incrementally elaborate an example model from abstract toward concrete implementation
- Discuss practical requirements imposed on the tool chain
- Engineers must make choices
  - Implementation choices should be minimally invasive and maximally deferred
  - Key questions underlying those choices…

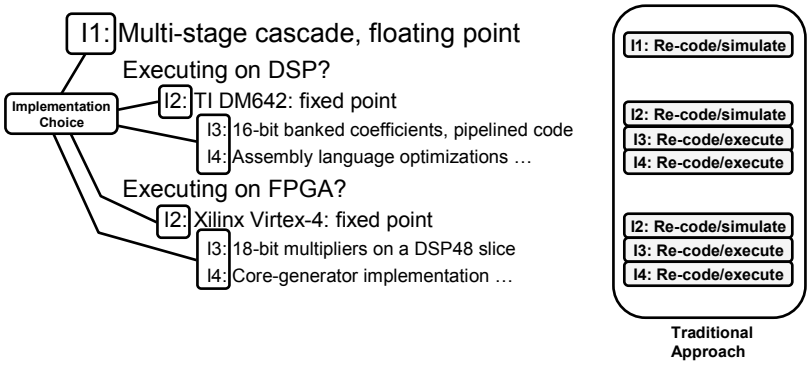
The MathWorks MATLAB&SIMULINK

Traditional Implementation Decisions

Abstract notion of an algorithm

Code/simulate

Example: Digital filter, prescribed specifications, floating point



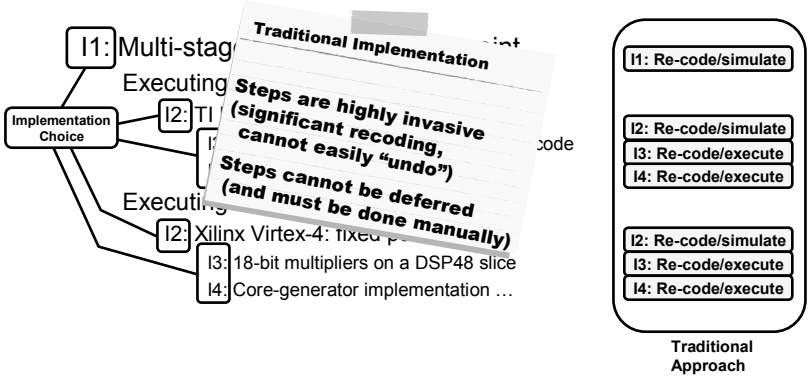
The MathWorks MATLAB&SIMULINK

Traditional Implementation Decisions

Abstract notion of an algorithm

Code/simulate

Example: Digital filter, prescribed specifications, floating point



The MathWorks MATLAB&SIMULINK

Traditional Implementation Decisions

- Additional Areas for Improvement
  - Performance scales with the engineer's experience (highly variable outcomes)
  - Time consumi… manual iterations
  - Initial design … all decision points
  - Code mainte… significant experience
  - Algorithm ch…
  - Design spa…
  - Verification… parately considered
  - Design art… iformly reproducible at each step
  - More…

Traditional Implementation

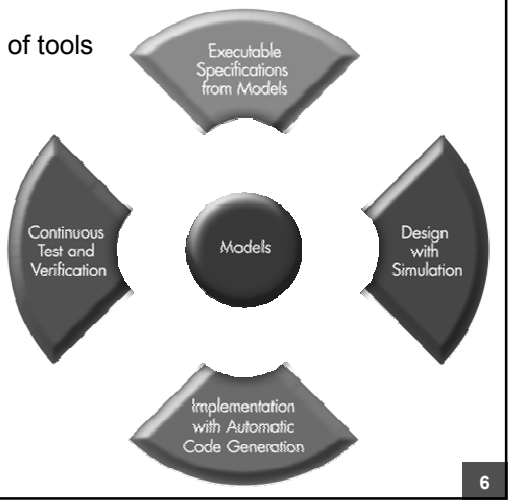
Steps are highly invasive (significant recoding, cannot easily "undo")

Steps cannot be deferred (and must be done manually)

The MathWorks MATLAB&SIMULINK

MBD Implementation Decisions

- Focus on the complete algorithm-design lifecycle
- MBD demands tight integration of tools
- Maintains model abstraction (defers implementation)
- Simplifies:
  - Understanding
  - Verification
  - Maintenance
  - Enhancement
  - Re-targetability



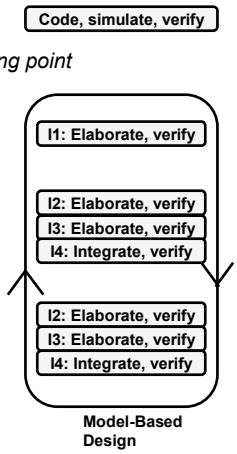
The MathWorks MATLAB&SIMULINK

MBD Implementation Decisions

Abstract notion of an algorithm

Example: Digital filter, prescribed specifications, floating point

- I1: Multi-stage cascade, floating point
  - Executing on DSP
    - I2: TI DM642, fixed point
      - I3: 16-bit banked coefficients, pipelined code
      - I4: Assembly language optimizations …
  - Executing on FPGA
    - I2: Xilinx Virtex-4, fixed point
      - I3: 18-bit multipliers on a DSP48 slice
      - I4: Core-generator implementation …

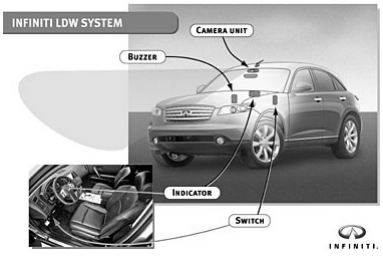


7

The MathWorks MATLAB&SIMULINK

Automotive Design Example: Lane Departure Warning

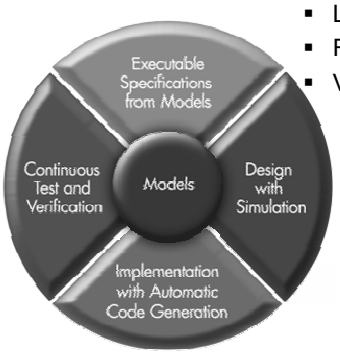
Results from Simulink Model of Lane Departure Warning System



8

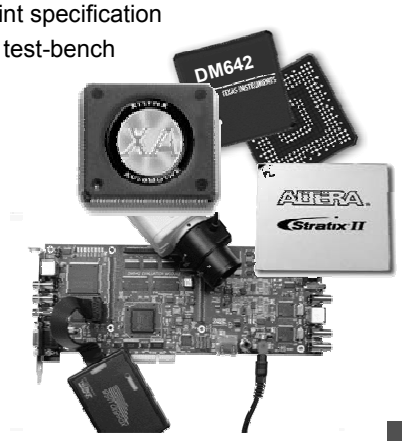
The MathWorks MATLAB&SIMULINK

MBD: Executable Specification



Executable Specifications

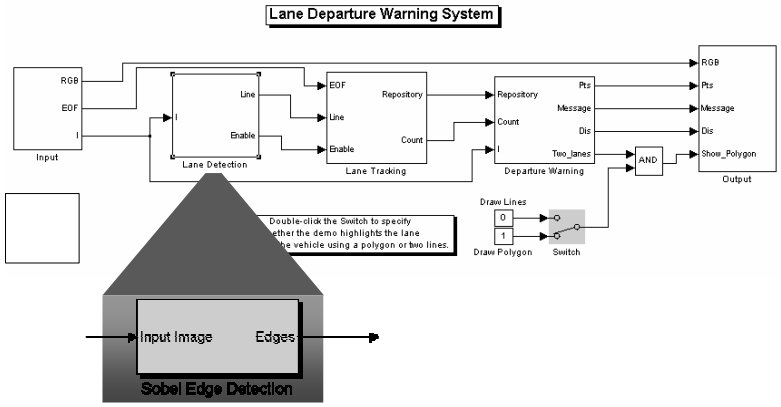
- Lane Departure Warning
- Floating-point specification
- Verification test-bench



The MathWorks MATLAB&SIMULINK

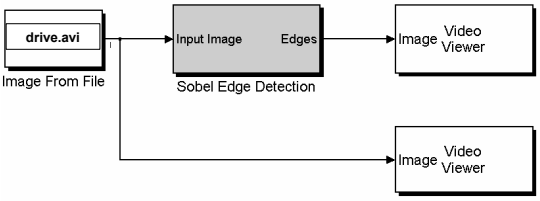
MBD: Executable Specification

System Model: Implement edge detection algorithm as part of lane departure warning system



Design Entry: Simulink

Start by developing a golden specification of the edge detection algorithm



Floating-point System Specification

High-level blocks: Video Edge Detector

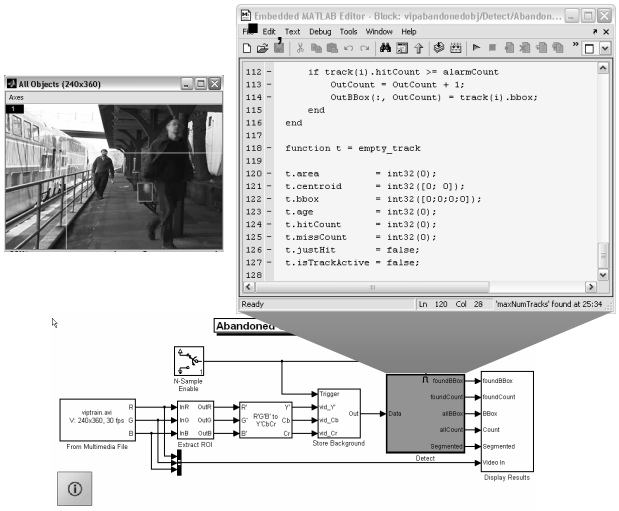
- Video and Image Processing Blockset

Lower-level blocks: Sum, Abs, 2-D Filter

- Simulink, Signal Processing Blockset

11

Design Entry: Embedded MATLAB



- Embedded subset of MATLAB Language
- Brings MATLAB algorithms into Simulink and Stateflow

12

The MathWorks MATLAB&SIMULINK

Insight on Implementation Choices

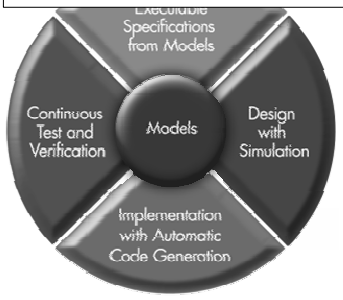
1. Design Languages
2.
3.
4.
5.

- **MATLAB:** Largely behavioral, un-timed, array-based semantics
- **Simulink:** Structural, time-based simulation engine, hybrid solvers
- **Stateflow:** Reactive systems design, finite state machines
- **SimEvents:** Discrete-event, data driven modeling

The MathWorks MATLAB&SIMULINK

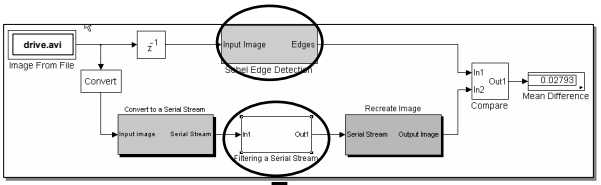
MBD: Design with Simulation

Design with Simulation
- Fixed-point model migration
- Design Advisors, Test-Benching

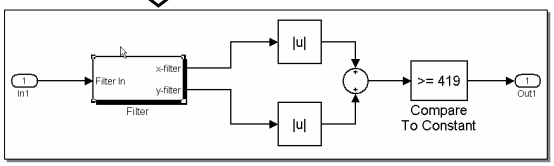


MBD: Design with Simulation

- Migrate the design using verified iterations
- Make decisions appropriate to deployment
  - Serial processing *(2D to vector data)*
  - Fixed-point operation *(float-to-fixed conversion)*



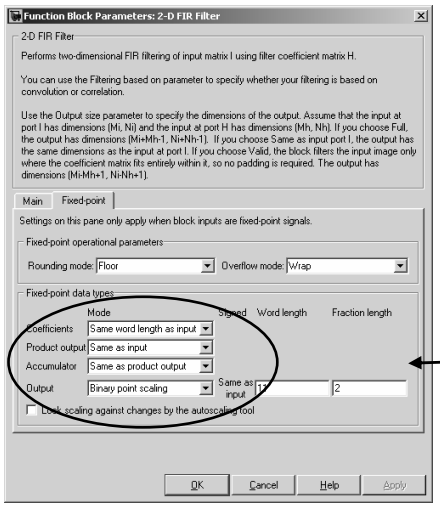
Floating-point model

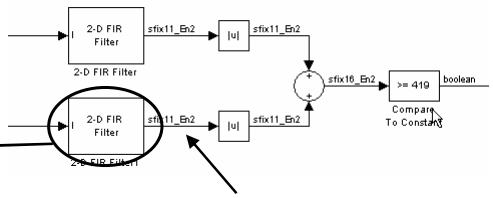


Fixed-point model

Migration to Fixed-point

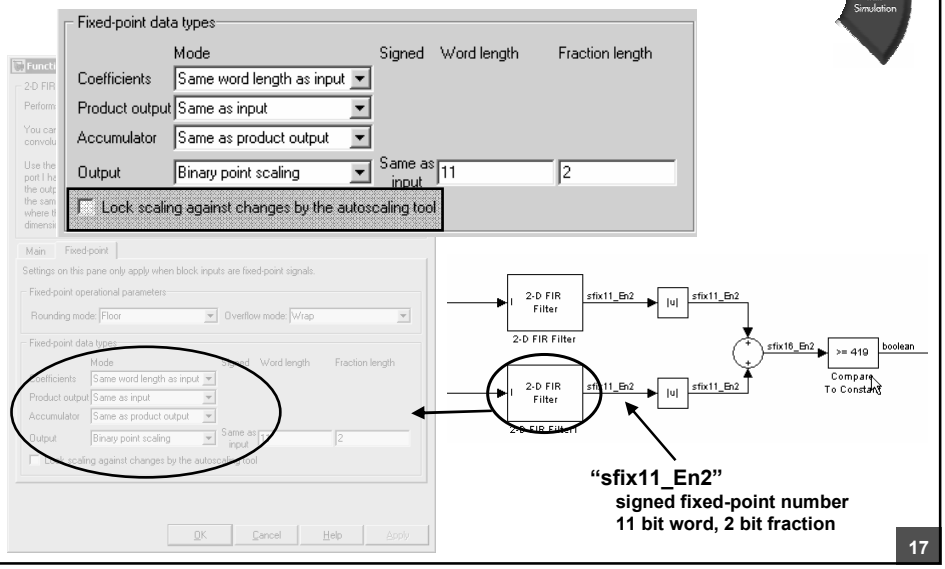
- Fixed-point data types – full manual control





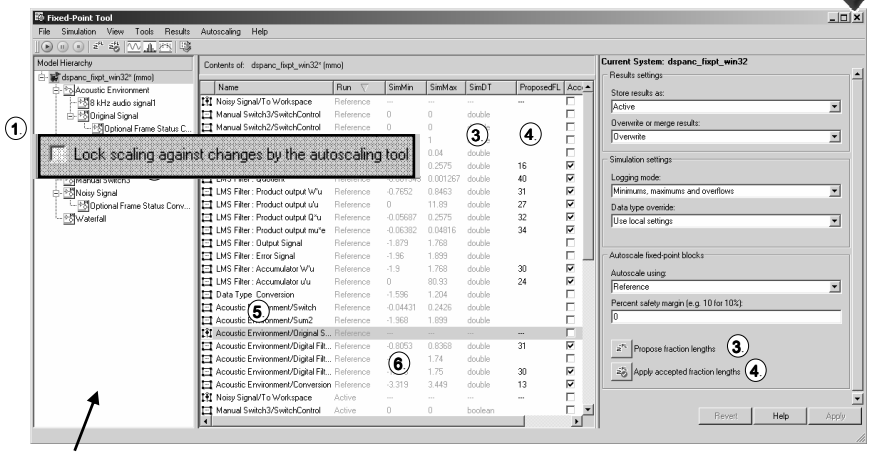
"sfix11_En2"
signed fixed-point number
11 bit word, 2 bit fraction

Migration to Fixed-point



Migration to Fixed-point

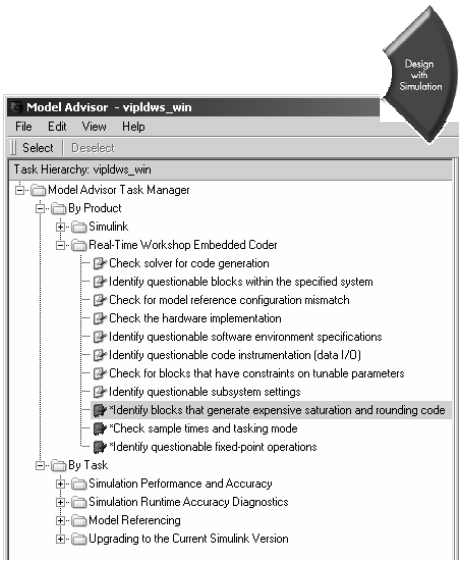
Autoscaling Tool: Advisor to help set fixed-point scale factors



The MathWorks MATLAB&SIMULINK

Design Advisors

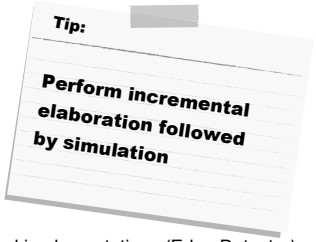
- GUI-driven Advice and Assistance
- Fixed-Point Tools
  - Record design requirements
  - Dynamic range assessment
  - Design auto-scaling
- Simulink Model Advisor
  - Choose advisor goals
    - Modeling Guidelines
    - Best Practices
    - Code Safety
    - Performance Optimization
  - Step-by-step guidance



The MathWorks MATLAB&SIMULINK

Insight on Implementation Choices

1. Design Languages
2. Model Elaboration
3.
4.
5.



Manual approach:

- Use block-based implementations (Edge Detector)
- Adjust fixed-point settings in each block dialog
- Use target-specific optimized blocks (TI, Xilinx, etc)

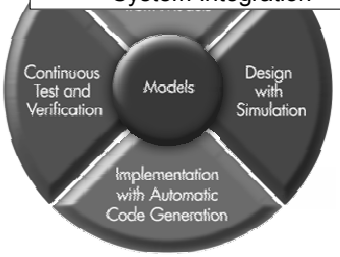
Assisted approach:

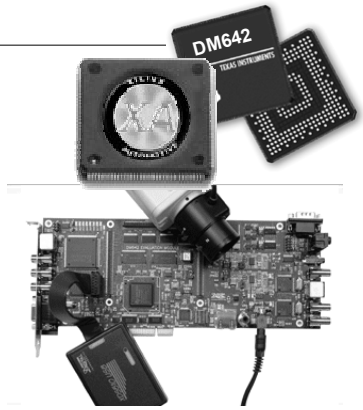
- Fixed Point Tool (Simulink Fixed Point)
  Automatically suggests fixed point settings for blocks
- Filter Wizard (Signal Processing Toolbox)
  Automatically generates Simulink models using basic blocks
- Simulink HDL Coder
  Generates new models based on implementation choices

The MathWorks MATLAB&SIMULINK

MBD: Automatic Code Generation

Automatic Code Generation
- Language Specialization (C, ASM, VHDL, Verilog)
- Design Space Exploration
- System Integration



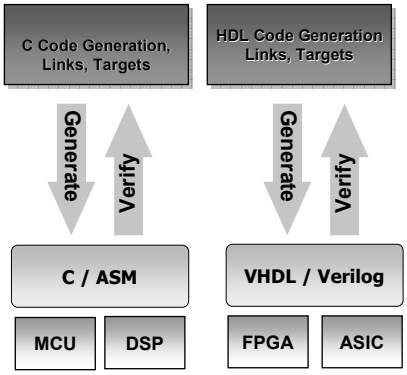


The MathWorks MATLAB&SIMULINK

MBD: Automatic Code Generation

Model-Based Design supports both Software and Hardware systems

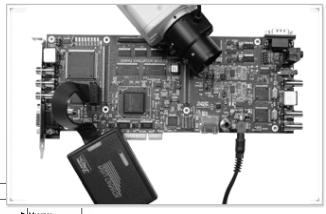
- Coders
  - Code generation from models
  - Language options
  - Code interfacing, optimization
- Links
  - Verification tool integration
  - Project generation, build, download
  - Co-simulation, SIL/PIL/HIL
- Targets
  - Processor & memory specific optimization
  - Device drivers, board support
  - Schedulers, RTOS integration

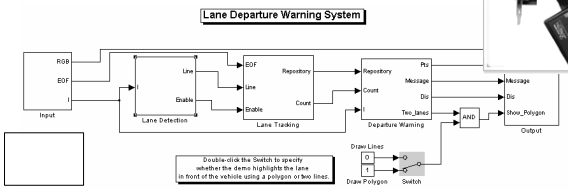


The MathWorks MATLAB&SIMULINK®

Code Gen: Embedded Software

- Code Generation
  - Real Time Workshop – ANSI/ISO C code for rapid prototyping, acceleration
  - Real Time Workshop Embedded Coder – Embedded deployment
- Links
  - Altium TASKING
  - Analog Devices VisualDSP++
  - Green Hills MULTI
  - TI Code Composer Studio
- Targets
  - TI C6000 DSP
  - TI C2000 DSP
  - Infineon C166 Microcontrollers
  - Freescale MPC5xx Microcontrollers



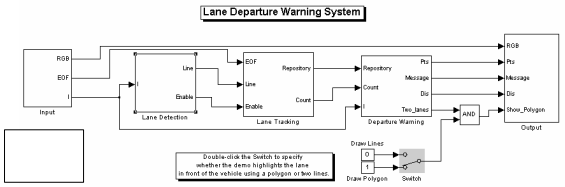


The MathWorks MATLAB&SIMULINK®

Code Gen: FPGA Hardware

- Code Generation
  - Simulink HDL Coder – FPGA and ASIC deployment using VHDL and Verilog
  - Filter Design HDL Coder – Filter implementation from MATLAB
- Links
  - Mentor ModelSim/Questa
  - Cadence VCS/Incisive
  - Synopsys Discovery
- Targets
  - Altera DSPBuilder
  - Xilinx SystemGenerator
  - More…

Let's take Lane Departure to an FPGA …



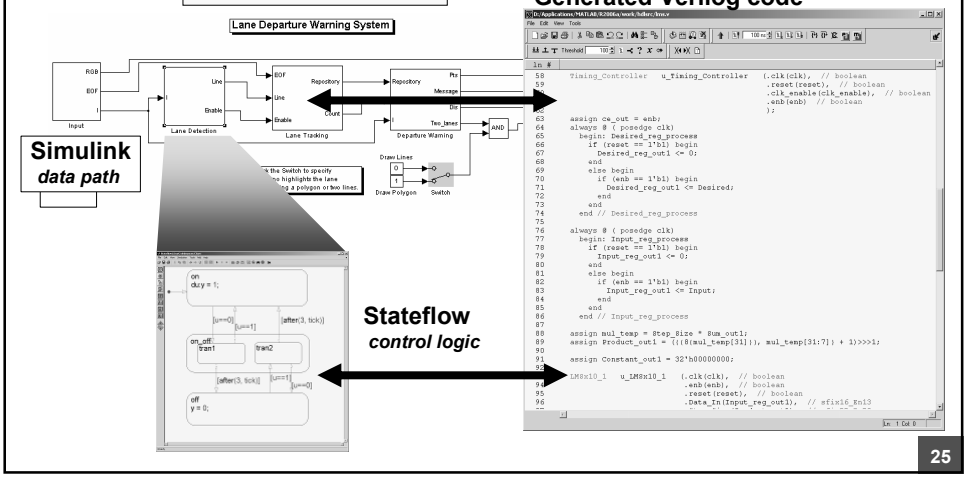
The MathWorks MATLAB&SIMULINK

Code Gen: FPGA Hardware

Implementation with Automatic Code Generation

Simulink HDL Coder
Correct-by-construction VHDL and Verilog code

Generated Verilog code



25

The MathWorks MATLAB&SIMULINK

Design Space Exploration for HDL

Implementation with Automatic Code Generation

NEED:

- **Speed**
  How fast can this design run?
- **Area**
  Can I use a smaller chip?
- **Power**
  Can I target a mobile device?

SOLUTION:

Code Generation Control Files

Controls implementation used to generate code for blocks / components

- **Locally:** for an individual block
- **Globally:** for all blocks of a given type
- **Hierarchically:** scope within a model

Sum, Product
- Linear
- Cascade
- Tree

Gain
- Multiplier
- CSD
- fCSD

Min/Max
- Tree
- Cascade

Lookup Table
- Inline
- Hierarchical

26

Example Control File

Textual interface

Initialize control file constraint object

CSD multiplier "GainCSD"

Factored-CSD multiplier "GainFCSD"

Editor - C:\WINNT\Profiles\don\Desktop\ldwsControlFile.m

File Edit Text Go Cell Tools Debug Desktop Window Help

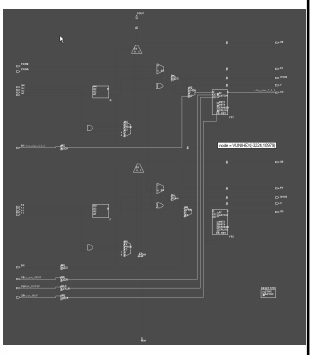
```
1  function c = ldwsControlFile
2   % Control file for HDL Code Generation
3
4   c = hdlnewcontrol(mfilename);  % new constraint object
5   c.set('TargetLang','VHDL');    % select VHDL code gen
6
7   % Generate code only for this subsystem:
8   c.generateHDLFor('ldws/LaneDetection');  % select subsystem
9
10  % Every gain block should use a "CSD" implementation
11  c.forEach('ldws/LaneDetection/*',...
12            'built-in/Gain', {},...
13            'hdldefaults.GainCSDHDLEmission');
14
15  % Except gain blocks corresponding to low clock rates
16  % For those we use "Factored CSD" (shorter delay paths)
17  c.forEach({'ldws/LaneDetection/Gain4',...
18             'ldws/LaneDetection/Gain5',...
19             'ldws/LaneDetection/Gain12',...
20             'ldws/LaneDetection/Gain13'},...
21            'built-in/Gain', {},...
22            'hdldefaults.GainFCSDHDLEmission');
```

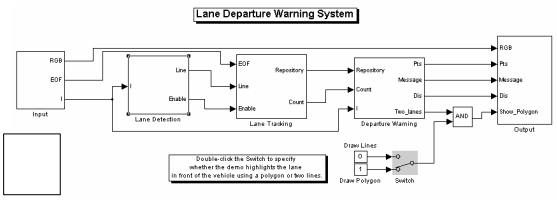
ldwsControlFile Ln 1 Col 1 OVR

Synthesis: Edge Detection


```
================================================================

Device utilization summary:
---------------------------

Selected Device : 4vsx25ff668-12

 Number of Slices:                     1105  out of  10240    10%
 Number of Slice Flip Flops:           1903  out of  20480     9%
 Number of 4 input LUTs:                156  out of  20480     0%
 Number of IOs:                          14
 Number of bonded IOBs:                  14  out of    320     4%
 Number of GCLKs:                         1  out of     32     3%

Design statistics:
    Minimum period:   4.106ns   (Maximum frequency: 243.546MHz)
    Minimum input required time before clock:   3.883ns
    Maximum output delay after clock:   7.323ns
```

XILINX®
VIRTEX-4 SX





Insight on Implementation Choices

1. Design Languages
2. Model Elaboration
3. Design Constraints
4.
5.

Code Gen Options
- Memory Layout
- Interfaces & Naming
- Clocking & Reset
- Pipelining Control
- Assertion Levels

Block Dialogs

Example block-level controls:

- Time- vs. Frequency domain algorithm options
- Table-lookup vs. on-line internal computations

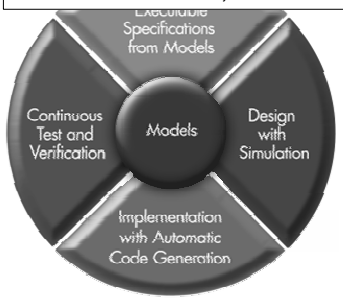
HDL Control Files

Specify concrete implementations
- For individual algorithms
- Across all similar algorithms

Re-simulate with constraints
- Bit-true, cycle-accurate
- Automatic model creation

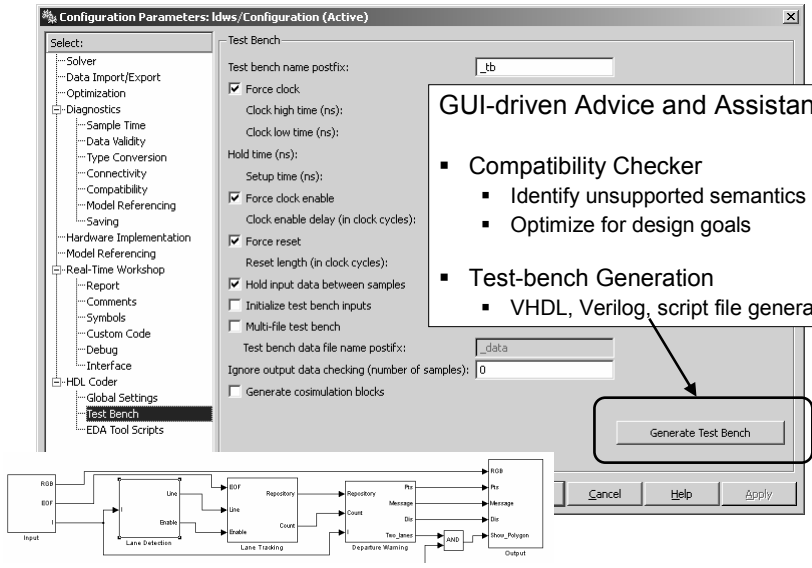
MBD: Continuous Test and Verification

Test and Verification
- Simulation, Test-benching
- SIL/PIL/HIL, Co-simulation



The MathWorks MATLAB&SIMULINK

Design Advisors for HDL

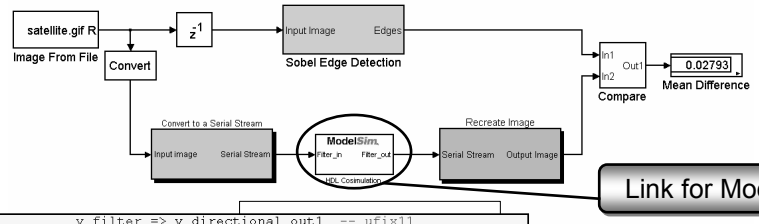


GUI-driven Advice and Assistance

- Compatibility Checker
  - Identify unsupported semantics
  - Optimize for design goals
- Test-bench Generation
  - VHDL, Verilog, script file generation

The MathWorks MATLAB&SIMULINK

Co-simulation of Generated Code



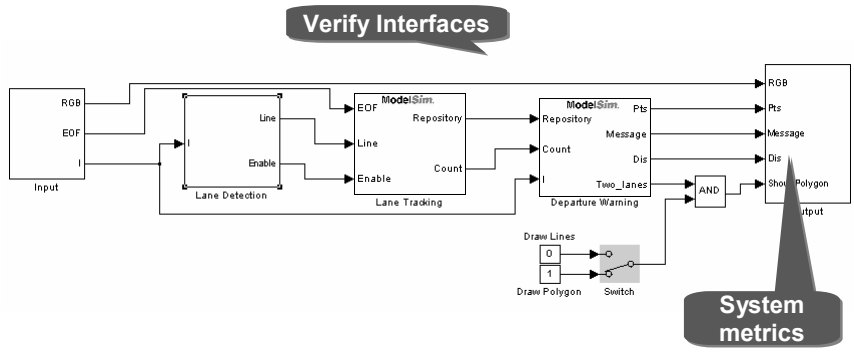
Link for ModelSim

```
117           y_filter => y_directional_out1  -- ufix11
118           );
119       s <= signed(filter_input);
120
121       Delay2_process : PROCESS (clk, r
122       BEGIN
123         IF reset = '1' THEN
124           Delay2_out1 <= (OTHERS => '0
125         ELSIF clk'event AND clk = '1'
126           IF enb = '1' THEN
127             Delay2_out1 <= s;
128           END IF;
129         END IF;
130       END PROCESS Delay2_process;
131
132
133       s_1 <= std_logic_vector(Delay2_o
134
```

HDL code executing on ModelSim simulator

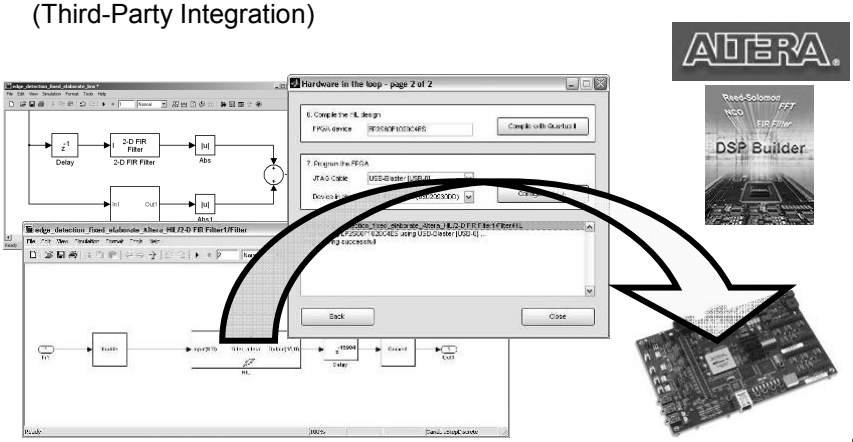
Making Full Use Of The System Model

- Promotes parallelism in design and verification tasks
- Accelerates system level verification



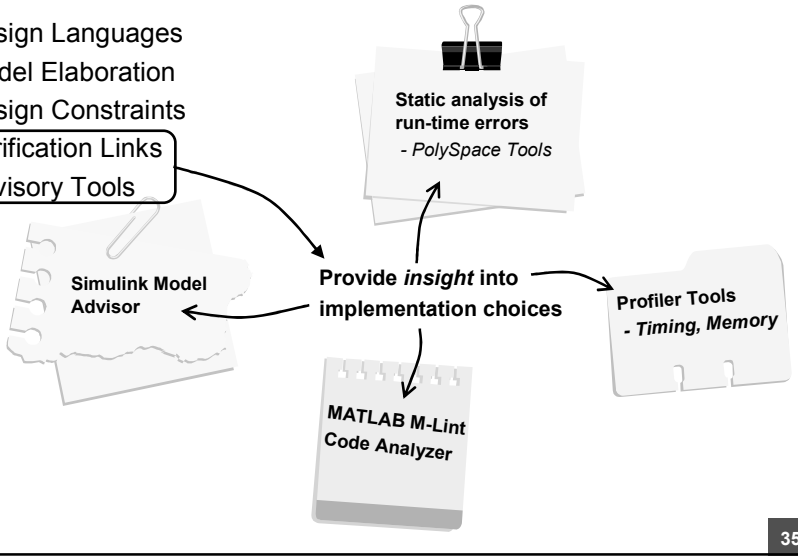
Verification of Synthesized Hardware

- Verify implementation using Hardware-In-the-Loop (Third-Party Integration)



Insight on Implementation Choices

1. Design Languages
2. Model Elaboration
3. Design Constraints
4. Verification Links
5. Advisory Tools



Insight on Implementation Choices

1. Design Languages
2. Model Elaboration
3. Design Constraints
4. Verification Links
5. Advisory Tools

Conclusions

- Maximally deferring implementation decisions demands Model-Based Design

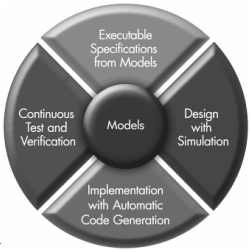
Conclusions?

- No single "silver bullet" to support modern workflow
  - Need a well-devised arsenal
- Well-defined and executable semantics are required
- Need integrated simulation, code generation, verification
- Multi-language support seems essential
  - Domain-specific, and combined textual & graphical

The MathWorks MATLAB&SIMULINK

Trends in Model-Based Design

- Unifying multiple engineering groups
  - One design environment
  - Multi-domain simulation (analog, digital, physical, …)
  - Multi-platform designs
    - Embedding MATLAB within Simulink and Stateflow
- Connecting stages of development
  - Design – Implementation – Verification
  - Coder products: C and HDL generation
  - Link products: Verification and Validation
- Harnessing high-performance computing
  - Distributed computing tools for MATLAB and Simulink
  - Acceleration from multi-core simulation



The MathWorks MATLAB&SIMULINK

Questions?

Accelerating the pace of discovery, innovation, development, and learning in engineering and science.

The MathWorks