



Model Based Design for FPGA Development

Charles Fulks


Intuitive Research and Technology
charles.fulks@irtc-hq.com
256-922-9300 Extension 129
www.irtc-hq.com

INTUITIVE
1ST DECADE
of excellence
EST. 1999



Abstract


- Model based design techniques save cost and schedule by verifying functionality and tuning system performance prior to logic design, simulation, integration, and test. This presentation demonstrates a simple mixed-signal programmable logic design through all design steps of model based design development. The demonstration includes model development, VHDL, testbench, Tcl scripts, and simulation. This presentation provides example files suitable for further development.



Prerequisites & Class Objectives

- Prerequisites:
 - A basic understanding of hardware descriptive languages and the associated design flow for developing digital circuitry
 - Previous experience with Matlab and Simulink (www.mathworks.com)
- Class objectives:
 - Discuss the benefits of model based design... and drawbacks
 - Demonstrate the model based design process applied to FPGA's
 - Focus on a contrived example that demonstrates the value of model based design in predicting system performance

3



Audience Composition

- How many of the audience have:
 - Prior Matlab / Simulink experience?
 - Fixed point tool experience?
 - Prior FPGA design experience?
 - Was the experience in Academia or in Industry?
 - Was the design capture implemented with Schematic capture or HDL
 - Verilog or VHDL
- Engineering focus software or hardware design?

4



Relevant Questions

- What are some of the common challenges associated with modeling a mixed system?
- What minimum set of information do I need to know to
 - start developing the model?
 - start implementing my logic design?
- What resources are available and which ones should I start with?
- What tools are available to help me develop real-world experience?
- How can I leverage my existing programming and digital design experience most effectively?


5



Agenda


- Background and Motivation
- Introduction to System Modeling
- What is a model?
- What is Model Based Design?
- Floating-Point and Fixed-Point
- Mathematical Approximations
- Model Based Design Example
- Conclusions

6



Model Based Design for FPGA Development


Background and Motivation



The Three Models

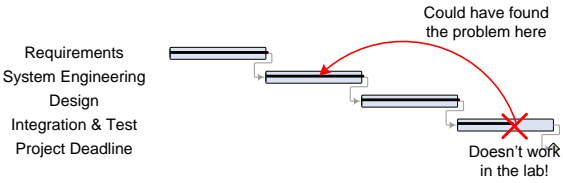
- System Model
 - System engineering trade space exploration
- Implementation Model
 - Includes target platform capabilities
- HDL
 - Describes the digital design

8




Project Motivation

Project ABC



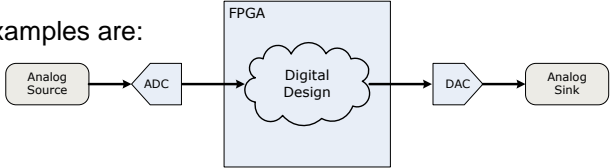
- Has this happened to your project?
- If the FPGA matches the model...
- ...and the model meets system requirements
- Then the FPGA will work in the system.

9



What is a “System”?

- In the context of modeling, the word system encompasses the entire product; not just a small portion such as the digital design
- In general, a real world system has many types of components
 - analog inputs (source)
 - analog outputs (sink)
 - digital circuits
 - analog-digital translation circuitry (ADC, DAC)
- Some analog IO examples are:
 - RF
 - Audio
 - Video

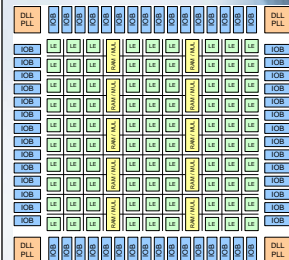


10



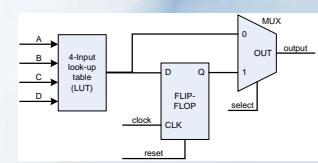
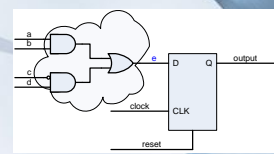
What is a FPGA?

- A Field Programmable Gate Array (FPGA) is a silicon chip (hardware)
 - It is a reconfigurable place to implement a digital design
- Think of it as a “blank page” of logic resources
 - Logic elements: LUT, register, adder
 - Multipliers: MAC, round, etc.
 - Memory: distributed and block
 - Input / output block: Multiple standards
 - Routing resources
- A FPGA provides a place for “virtual hardware”
 - It allows a developer to tailor the computer architecture to the numerical problem for maximum performance



What is a FPGA?

- How do you “reconfigure” logic gates?
 - Placing discrete gates to realize a logic function can create very fast circuits in the smallest area (think ASIC)
 - ...but it is not an efficient way to make things that are reprogrammable.
 - And costs a lot for each change
- Any arbitrary 4 input logic function may be implemented with a 16-bit memory look-up table
 - A, B, C, and D form the address
 - Storing the 16 combinations can form any arbitrary 4-input logic function
 - Note SRAM keeps values only while power is applied





Some Typical FPGA Applications

- Processor
 - ARM, ColdFire, MicroBlaze, Nios II, PPC, others
- I/O Expansion
 - control signal distribution
 - address decoding
- Peripherals
 - Communication: SPI, I2C, RS-232, Ethernet, PCI
 - ADC, DAC
 - PWM
 - Motor control: stepper, DC, BLDC, AC
- Direct digital synthesis
 - sine, cosine, arbitrary waveform
- Interface bridging

13




What is HDL?

- Hardware Description Language (HDL)
 - VHDL, Verilog
- VHDL is...
 - Hardware: It can be used to describe digital designs
 - Software: It can be used to describe non-synthesizable algorithms such as file parsing routines in a test bench
- MIT Open Course Ware, Complex Digital Systems, Spring 2005, Lecture 2: Digital Design Using Verilog:
 - An HDL is NOT a Software Programming Language
 - Software Programming Language: Language which can be translated into machine instructions and then executed on a computer
 - Hardware Description Language: Language with syntactic and semantic support for modeling the temporal behavior and spatial structure of hardware

HDL's are not software programming languages


14



Why Model Based Design

- The goal is to produce a digital design for a FPGA that will meet system requirements
- Designing a signal processing algorithm in HDL
 - is slow (expensive)
 - is error prone (expensive)
 - may not be possible
- Model based design looks at the system as a whole
 - Avoids a myopic view of the FPGA
- Determining the correct signal processing algorithm prior to HDL capture
 - takes more time up front, but is faster overall (cheaper)
 - finds errors sooner in the design cycle (cheaper)
 - almost always feasible

15



Model Based Design for FPGA Development

Introduction to System Modeling

16



What is Model Based Design?

- A method of modeling the design implementation as it will run on the target to eliminate errors early in the design cycle.
- System performance is expected to match the model performance
 - The model is used as the basis of comparison for unit level testing
 - The model becomes the primary tool in trouble shooting system failures


17



What is a Model?

- A model mathematically represents a system
 - More fidelity provides better information, but takes more time
 - Enables a high degree of understanding regarding system behavior
- Modeling blurs the distinction between HW and SW engineers
 - Shift toward domain expertise


18



Implementation Model

- An implementation model contains:
 - Complete target specific design specifications
 - signal range and resolution specifications
 - Realizable algorithm structures; causal
 - Approximation algorithms (due to target resource trade-offs)
 - sin/cos table look-ups, arctan, sqrt, etc.
- An implementation model accurately simulates:
 - Algorithm behavior as implemented on the target
 - The target numerical limitations
 - fixed point operations when the target is fixed point
 - Signal timing and latency
 - Actual system performance


19



What does the Model do for us?


- The implementation model proves that an algorithm will or will not meet requirements
 - Provide real world input stimulus (measured or simulated) to the model and examine the model output
 - Sometimes called an “Executable Specification”
- Ensures the signal processing algorithm meets system requirements prior to:
 - FPGA design
 - Integration and test
- Saves time and money

20

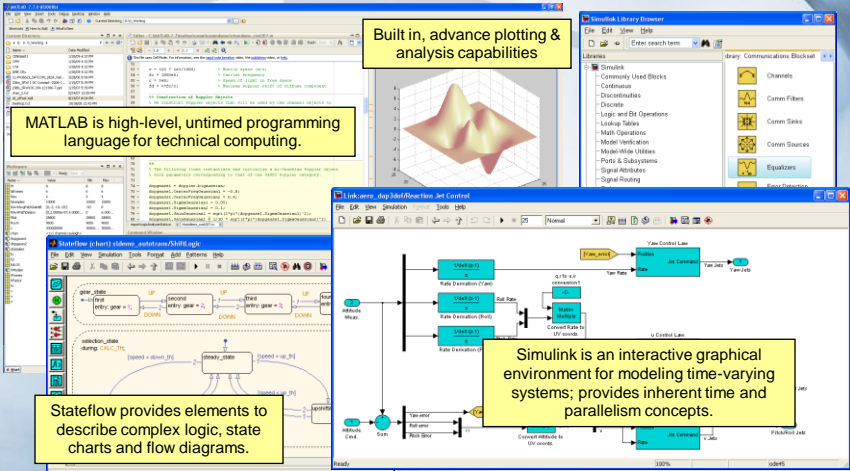


Modeling Tools

- To model systems containing FPGA's we need a tool that can produce a bit accurate fixed-point model
 - Many tools are available such as C / C++, Matlab, Simulink, MatrixX / LabView
- Automated HDL generation tools are available which generate target independent, bit-true, cycle-accurate, synthesizable Verilog and VHDL code from Simulink models
 - Simulink HDL Coder (The Mathworks, Inc.)
 - Synplify DSP (Synplicity, Inc.)
- Similar automated HDL generation tools produce target dependent HDL from Simulink models
 - Altera DSP Builder
 - Xilinx System Generator



Matlab/Simulink




MATLAB is high-level, untimed programming language for technical computing.

Built in, advance plotting & analysis capabilities

Stateflow provides elements to describe complex logic, state charts and flow diagrams.


Simulink is an interactive graphical environment for modeling time-varying systems; provides inherent time and parallelism concepts.



Model Based Design for FPGA Development

Floating-Point and Fixed-Point

23



Model Development

- In today's complex signal processing systems, modeling is almost always used by systems engineering to analyze and specify system design and performance.
 - Trade studies and analysis of various algorithms
 - Rapid "prototyping" via generic algorithm libraries (e.g. Simulink)
 - Parametric analysis, e.g. Monte Carlo, etc.
- Systems engineering designs are done almost exclusively in floating point models.
 - These models become basis of the system specifications
 - *They require translation when the target is a fixed point platform*

24



Translation to Fixed Point

- Requirements to translate a generic model to a fixed point target
 - Fixed point signal specifications
 - Range and resolution; number of bits, number of fractional bits
 - Saturate or wrap; describes overflow behavior
 - Truncate or round; select required bits from intermediate results
 - Resource efficient processing
 - Algorithm computations cannot exceed resource allocation
 - Approximations for math functions (e.g. sin/cos, sqrt, arctan, etc.)
- Memory and buffering to maintain causal operations


25



Effects of design translation

- Design translation introduces:
 - degradation/error due to choices of range/resolution, truncation, etc.
 - processing degradation/error due to use of approximations for math functions such as sin/cos, sqrt, arctan, etc.
 - latency and timing effects due to buffering, etc.
- If these effects are not modeled they will introduce uncertainty into the system performance
 - The system performance fails to meet specifications
 - Labor intensive system troubleshooting
 - Mismatch between the model and the implementation
 - The model loses value for unit testing the implementation
 - The model loses value for troubleshooting lab issues
 - The implementation and integration exceed budget and schedule


26



Fixed point vs. floating point models

- Floating Point Models
 - Pros
 - Design speed
 - numeric limitations can generally be disregarded
 - algorithm libraries are readily available – rapid prototyping
 - Simulation speed
 - Floating point simulations run much faster
 - Cons
 - *Provides no insight* or guidance regarding numeric limitations of fixed point operations and their affect on system performance
- Fixed Point Models
 - Pros
 - Forces consideration and specification of fixed point operations
 - Actual system performance can be analyzed
 - Cons
 - Design speed
 - greater algorithm design effort
 - Simulation speed
 - less effective for system design

27




Costs of Implementation Modeling

- Additional Development Time
 - The implementation will get designed; the questions are:
 - How effectively?
 - How accurately?

Without a model we don't know until late in the design.
- Design Iteration Issues
 - The project may require maintenance of two models as the design iterates
 - Efficient systems design may still require use of a floating point model for some analysis and simulation intensive tasks

28




Benefits of Implementation Modeling

- System performance verified BEFORE implementation and test
 - System performance should match the model performance
- Complete design specifications
 - Rapid implementation and integration
- Logical and functional flaws found during design phase
 - Early detection is MUCH cheaper to fix
- The implementation model provides a
 - unit test benchmark
 - highly effective debugging tool

The cost and schedule savings far outweigh the cost of additional design time!

29



Model Based Design for FPGA Development

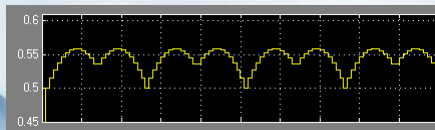
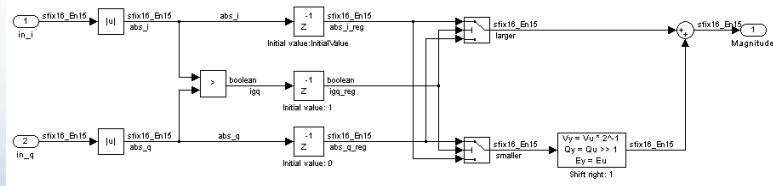
Mathematical Approximations

30

- Consider a system that requires calculation of the magnitude of a complex number; $(r^2 + i^2)^{0.5}$
 - The squaring operations each require a multiplier
 - The square root operation requires significant time and resources (clock cycles and circuitry)
- An approximation that meets system requirements is a better solution
 - Note the approximation will introduce error to the data path. The model will prove whether the error is acceptable.

31

- Complex magnitude approximation
 $(r^2 + i^2)^{0.5} = \text{larger}(|r|, |i|) + 0.5 * \text{smaller}(|r|, |i|)$

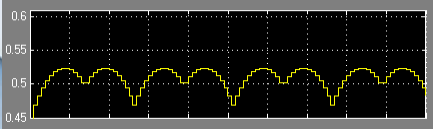
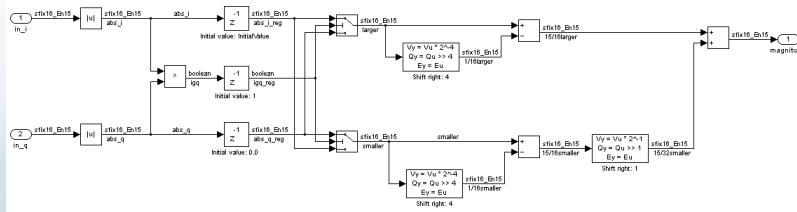


32



Math Approximation Example

- More accurate approximation
 $(r^2 + i^2)^{0.5} = (15/16)\text{larger}(|r|,|i|) + (15/32)\text{smaller}(|r|,|i|)$
 Note these fractions are shift and add operations

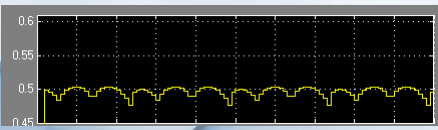
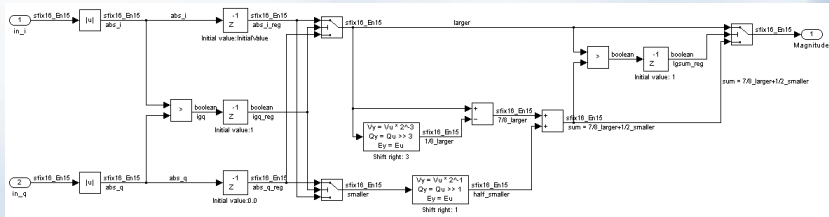


33




Math Approximation Example

- More accurate approximation
 $(r^2 + i^2)^{0.5} = \max[\text{larger}, (7/8)\text{larger} + (1/2)\text{smaller}]$



34



Math Approximation Take-away

- Changing algorithms during HDL capture is prohibitively expensive
- Changing algorithms during modeling is cost effective

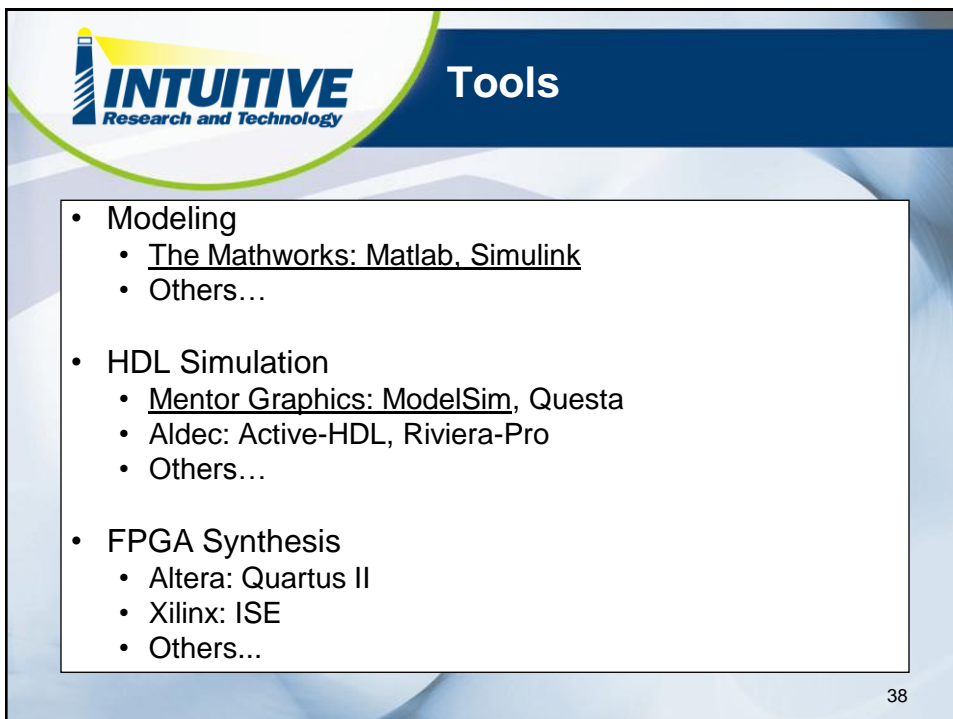
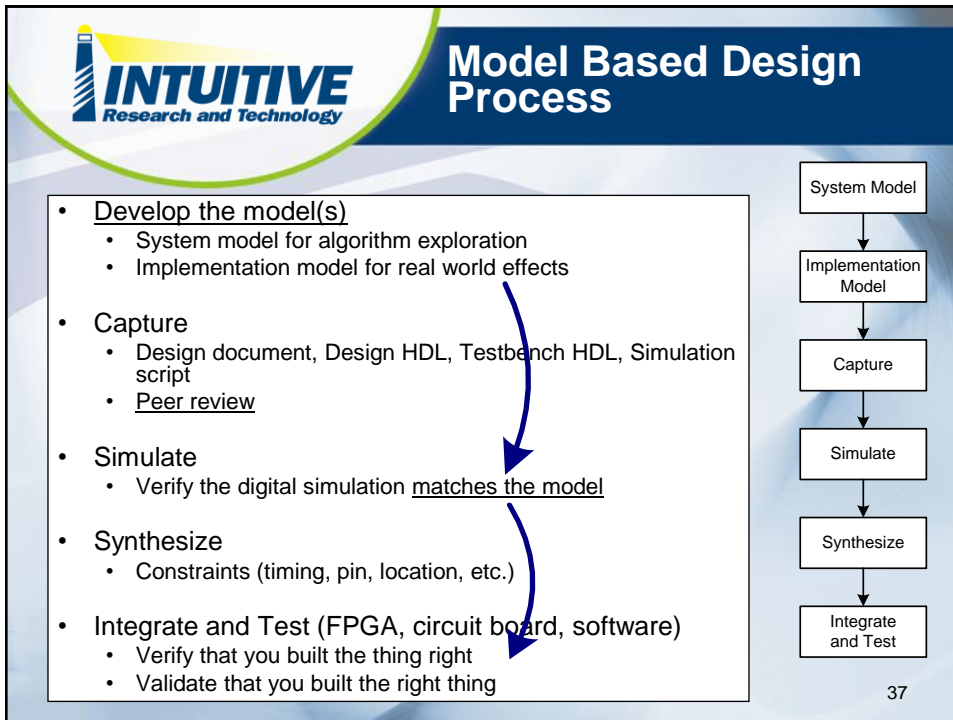
35




Model Based Design for FPGA Development

Model Based Design Example

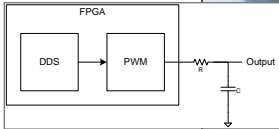
36






Requirements

- Requirement: A low frequency analog output (light, sound, or motion)
- Constraints: No digital-to-analog converter
- Design idea:
- Pulse width modulation (PWM) circuit driving a R-C low pass filter
- Implications:
 - Digital FPGA circuit closely tied to an external analog circuit
 - Impossible to gauge the “correctness” of the digital circuit without a model of the system (or the actual system)
- Note: Altera WP-01085-1.0, November 2008, “Controlling Analog Output From a Digital CPLD Using PWM”




39



Model Based Design for FPGA Development

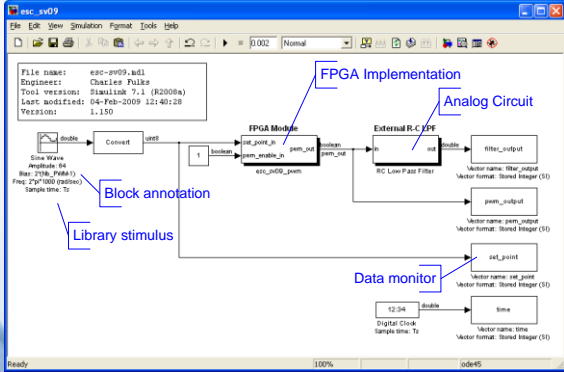
System Model Development

40




System Model

- Uses library block stimulus for quick modeling

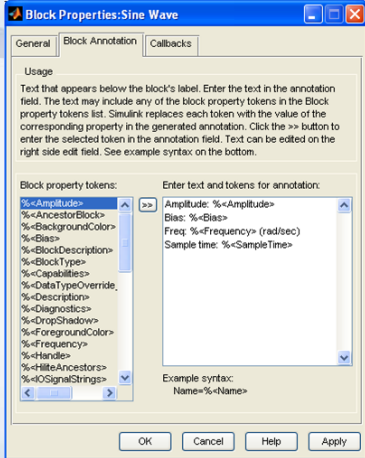


41



Block Annotations

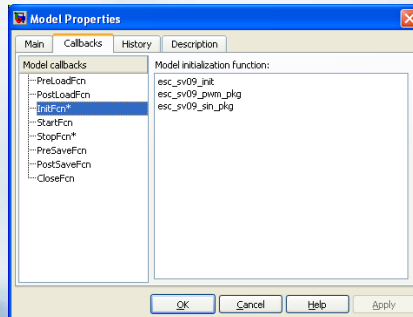
- Extremely helpful in understanding the model
- Right click on the block and select Block Annotations



42

Model Callbacks

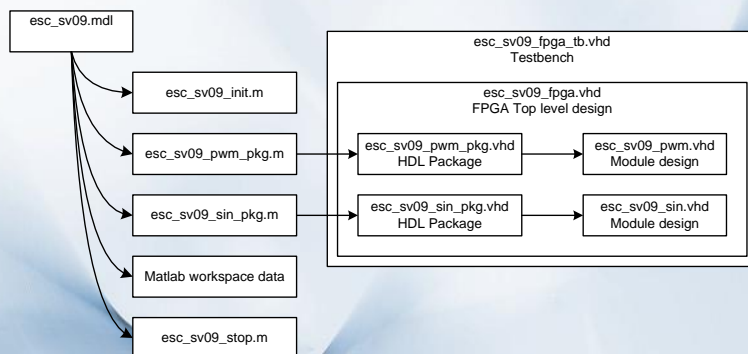
- Used to run Matlab scripts (m-files) at different points in the simulation
- Right click on the model and select Model Properties



43

Model / HDL File Relationships

- Use m-files to generate HDL to tie the model to the FPGA design



44



Model initialization script: init.m

- Set up all parameters used by both the model and the implementation

```
clear; % Clear all variables in the workspace
clc; % Clear the command window

tic; % start a simulation timer

Fs = 100e6; % Sample frequency
Ts = 1/Fs; % Sample time
Nb_PhAcc = 24; % Number of bits in the sin wave phase accumulator
Nb_PWM = 8; % Number of bits in the PWM free-running counter

% counter roll-over frequency
Fctr = 1/(Ts*(2^Nb_PWM));
fprintf('The PWM counter frequency is %6.2f Hz\n', Fctr)

Fco = 2500; % Analog R-C low pass filter
R = 100;
C = 1 / (2*pi*Fco*R);
fprintf('For Fco = %d and R = %6.2f, C = %f uF\n', Fco, R, C*10^6)
```

45



PWM HDL Generation pwm_pkg.m

- Automate the header

```
filename = 'esc_sv09_pwm_pkg.vhd';
fid = fopen(filename, 'w');
if fid == -1
    fprintf('File error!\n')
else
    time_vec = fix(clock); % get the system date and time
    hour = time_vec(4);
    min = time_vec(5);
    % Write the header to the file
    fprintf(fid, '-----\n');
    fprintf(fid, '-- Company: Intuitive Research and Technology\n');
    fprintf(fid, '-- Engineer: Charles Fulks\n');
    fprintf(fid, '-- File name: %s\n', filename);
    fprintf(fid, '-- Description:\n');
    fprintf(fid, '-- \n');
    fprintf(fid, '-- This file contain constants used in the PWM module\n');
    fprintf(fid, '-- \n');
    fprintf(fid, '-- Revision: \n');
    fprintf(fid, '-- \n');
    fprintf(fid, '-- %s, %d:%02d Written by esc_sv09_pwm_pkg.m\n', date, hour, min);
    fprintf(fid, '-- \n');
    fprintf(fid, '-- DO NOT MODIFY THIS FILE!\n');
    fprintf(fid, '-- This file was generated by the MatLab script esc_sv09_pwm_pkg.m\n');
```

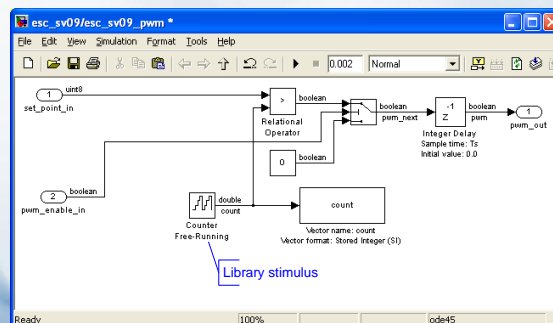
46

- Writing HDL from scripts called and used by the model is a way to ensure the HDL matches the model *exactly*

```
fprintf(fid, 'library ieee;\n');
fprintf(fid, 'use ieee.std_logic_1164.all;\n');
fprintf(fid, '\n');
fprintf(fid, 'package esc_sv09_pwm_pkg is\n');
fprintf(fid, '\n');
fprintf(fid, '    constant Nb_PWM      : natural := %d;\n', Nb_PWM);
fprintf(fid, '\n');
fprintf(fid, 'end package esc_sv09_pwm_pkg;\n');
fclose('all');
fprintf('Finished writing vhdl package.\n\n')
end
```

47

- A counter from the Simulink library is used to generate the PWM signal



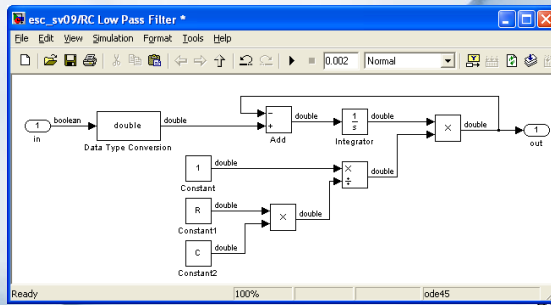
48

- RC Low pass filter
 - Implemented with a resistor and capacitor connected to the FPGA output pin

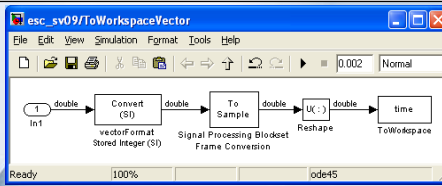
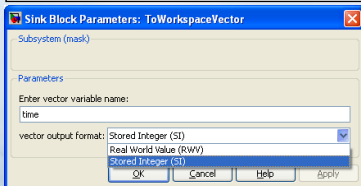
$$Fco = \frac{1}{2\pi RC}$$

$$\frac{V_{in} - V_{out}}{R} = C \frac{dV_{out}}{dt}$$

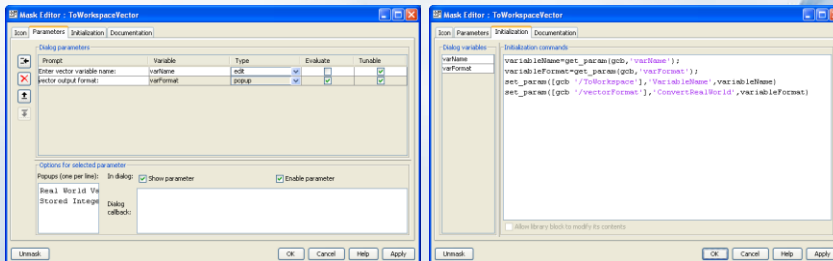
$$V_{out} = \int (V_{in} - V_{out}) dt$$



- Stores the simulation data to a Matlab workspace variable for later use
 - Can store to a file for use in HDL testbench
- This is a masked subsystem
 - Right-click and select Look under mask



- Masked subsystem
 - Right-click and select edit mask



51

- Actions to perform after simulation is complete
 - Plot simulation results
 - Store data to files
 - Etc.

```

step_time = 0.0007;
delta = 3/Fcptr;

figure(gcf); % get current figure

subplot(2,1,1);
plot(time, pwm_output, time, count/(2^Nb_PWM), ':');
axis([step_time-delta step_time+delta -0.1 1.1]); %[XMIN XMAX YMIN YMAX]
title('PWM Output');
xlabel('time (sec)')
grid on

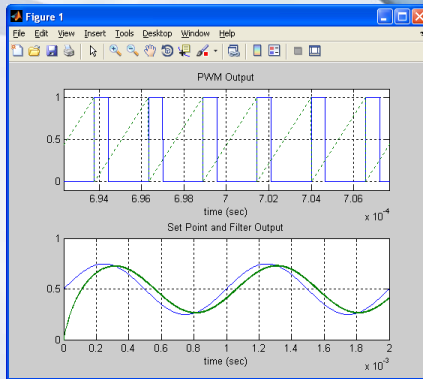
subplot(2,1,2);
plot(time, set_point/(2^Nb_PWM), time, filter_output);
axis([-inf inf 0 1]); %[XMIN XMAX YMIN YMAX]
title('Set Point and Filter Output');
xlabel('time (sec)')
grid on

toc % stop the timer and display result

```

52

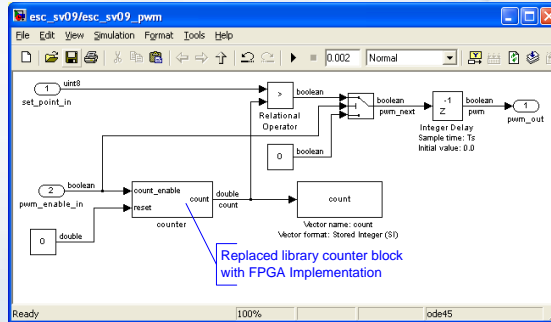
- The top plot shows the counter and PWM waveform
- The bottom plot shows the FPGA output and the R-C filter output
 - Note the initial condition as well as the amplitude and phase artifacts



53

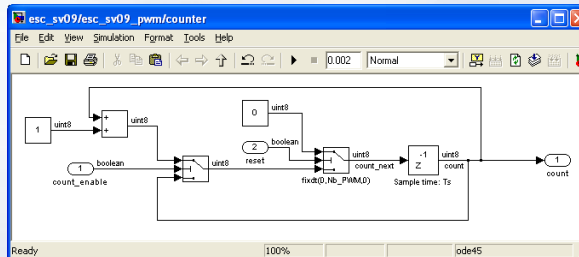
Implementation Model Development

54



55

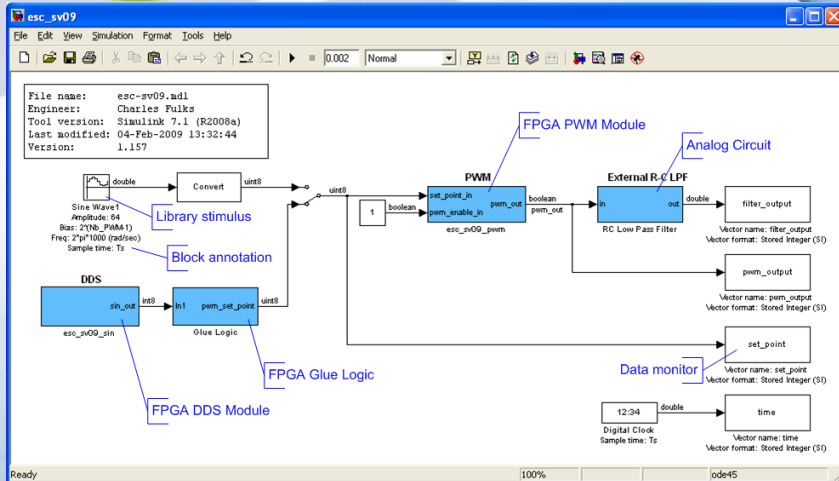
- Fixed point operations sometimes result in unexpected warnings



Warning: Overflow condition has been detected at time 2.55e-006 in 'esc_sv09/esc_sv09_pwm/counter/Add' which has resulted in an undefined operation. The results of this simulation are hardware dependent and may not be reproducible on other computers running Simulink. Consider correcting this condition by setting 'Saturate on integer overflow' parameter to 'on'. Suppressing additional overflow warnings and continuing simulation.

56

Add DDS Stimulus



57

DDS HDL Generation sin_pkg.m


- We need a sine wave table for use in the model and the FPGA

```
filename = 'esc_sv09_sin_pkg.vhd';

TableLength = 64; % entries
TableAddrWidth = log(TableLength)/log(2)-2;
TableDataWidth = 7; % bits
TableScale = 2^(TableDataWidth-2);
DT = 2^(-TableAddrWidth-2); %Table increment
%initialize vector for use in the model
SinTable = zeros(1,TableLength);

% Write the header to the file
```

58




DDS HDL Generation

sin_pkg.m

```

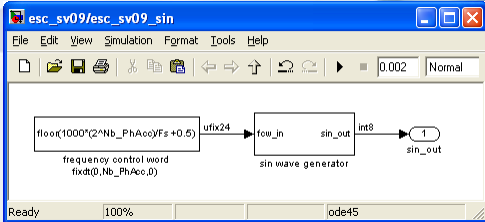
fprintf(fid, 'subtype table_address_type is integer range 0 to TABLE_LENGTH-1;\n');
fprintf(fid, 'trig_array_type is array(table_address_type) of integer;\n');
fprintf(fid, '\n');
fprintf('Writing SinTable ROM values...\n');
fprintf(fid, '    constant sin_array : trig_array_type := (\n');
fprintf(fid, '    -- addr : decimal value\n');
for k=0:TableLength-2
    SinTable(k+1) = floor(TableScale*sin(2*pi/4*DT*k));
    fprintf(fid, '%3d -- %4d : %1.8f\n', SinTable(k+1), k, SinTable(k+1)/TableScale);
end
% create the last table entry (VHDL difference)
k = TableLength-1;
SinTable(k+1) = floor(TableScale*sin(2*pi/4*DT*k));
fprintf(fid, '%3d -- %4d : %1.8f\n', SinTable(k+1), k, SinTable(k+1)/TableScale);
fprintf(fid, '    );\n\n');
fprintf(fid, 'end package esc_sv09_sin_pkg;\n');
fclose('all');
```

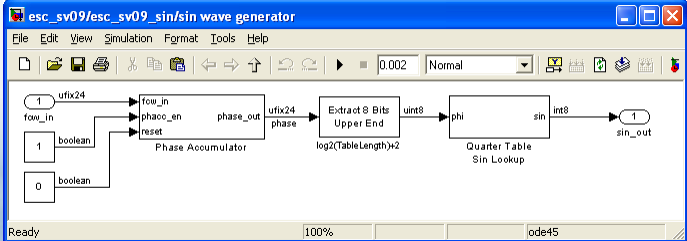
59



DDS Implementation Model

- FCW Calculation results in count by 168



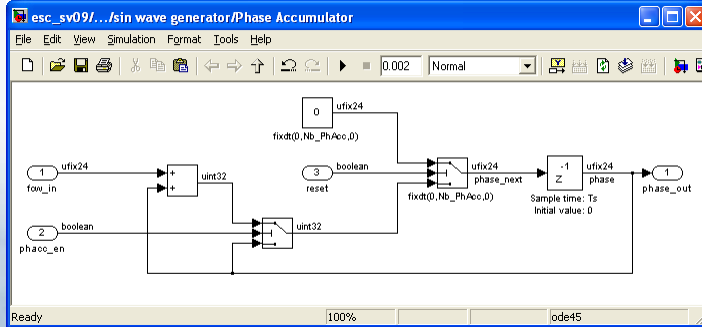


60



Phase Accumulator

- Counter counts by 1
- Phase accumulator counts by FCW

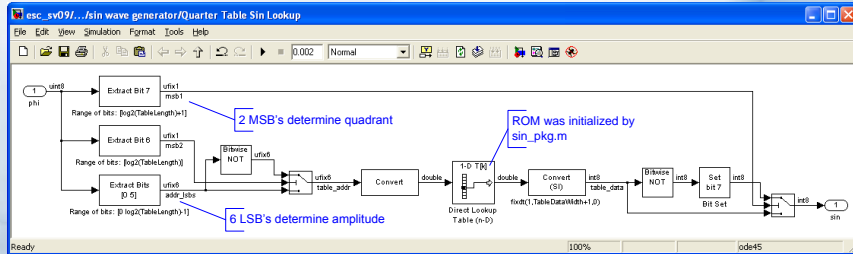


61



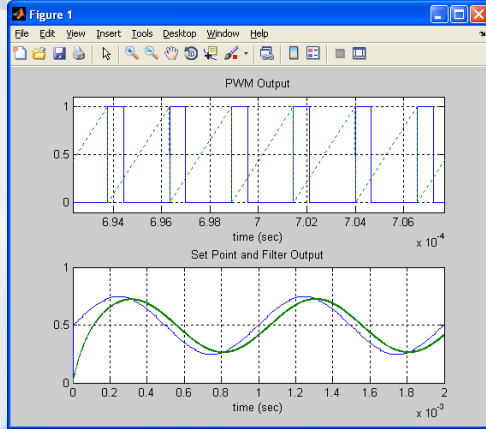
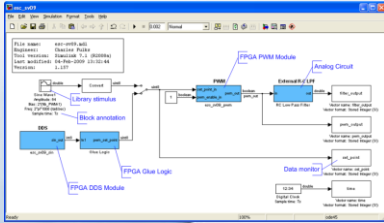
Quarter Table Sin Function

- Uses quarter wave symmetry
 - Store quadrant 1 in ROM
 - Use 2 MSB's to reflect quarter wave to other quadrants



62

- Verifies that the model meets requirements



FPGA Implementation



Simulation script

- A script ensures the tools run in a consistent manner
 - Compilation
 - ensures the library is up to date
 - design entities
 - testbench entities
 - Simulation
 - places signals of interest

65



Simulation script: fpga.do

- The header saves time by reducing confusion

```
# File name:      esc_sv09_fpga.do
# Target Devices: Simulation only
# Tool versions: ModelSim 6.3c
# Description:
#
# This file is the TCL script for ModelSim that performs the following.
# - compile the design with Lint checking
# - compile the testbench with Lint checking
# - start the simulation
# - add formatted signals to the wave window
# - run the simulation for a predetermined amount of time
#
# Dependencies:
# esc_sv09_sin_vhdl_pkg.vhd, esc_sv09_pwm_pkg.vhd, esc_sv09_fpga.vhd
# tb_utilities_pkg.vhd, esc_sv09_fpga_tb.vhd
#
# Revision:
# 15-Dec-2008 cf Created.
# 2-Feb-2009 cf Peer reviewed with Brian Morris
# 2-Feb-2009 cf Simulated
```

66



Simulation script: fpga.do

- Compilation ensures the library is up to date
- Lint finds errors fast

```
set simulate "yes"
# Set how long to run the simulation
set run_time 1.lms
# set directory aliases
set script_dir "Z:/My Documents/esc-sv09"
set testbench_dir "Z:/My Documents/esc-sv09"
set source_dir "Z:/My Documents/esc-sv09"
# create a design library
vlib work
# compile the design
echo "Compiling the design..."
vcom -2002 -check_synthesis -lint $source_dir/esc_sv09_sin_pkg.vhd
vcom -2002 -check_synthesis -lint $source_dir/esc_sv09_sin.vhd
vcom -2002 -check_synthesis -lint $source_dir/esc_sv09_pwm_pkg.vhd
vcom -2002 -check_synthesis -lint $source_dir/esc_sv09_pwm.vhd
vcom -2002 -check_synthesis -lint $source_dir/esc_sv09_fpga.vhd
# compile the testbench
echo "Compiling the testbench"
vcom -2002 -lint $testbench_dir/tb_utilities_pkg.vhd
vcom -2002 -lint $testbench_dir/esc_sv09_fpga_tb.vhd
```

67



Simulation script: fpga.do

- This is an excerpt from the script

```
if {$simulate == "yes"} {
# start the simulation
# vsim -t <resolution> <entity> <architecture>
vsim -t 10ps esc_sv09_fpga_tb esc_sv09_fpga_tb_arch

# add signals to the wave window
add wave -divider {DUT Interface}
add wave DUT/reset
add wave DUT/clk

add wave -expand -group Sin
add wave -group Sin -radix decimal DUT/inst_esc_sv09_sin/fcw_in
add wave -group Sin -format Analog-Step -height 50 -offset 0 -radix unsigned -scale 2.74e-006 \
/esc_sv09_fpga_tb/dut/inst_esc_sv09_sin/phase

add wave DUT/pwm_out

# run the simulation for the specified amount of time
run $run_time
wave zoomfull
}
```

68



Testbench

- Instantiates the Device Under Test (DUT)
- Provides stimulus and may record DUT behavior
- Required in some form for any simulation

69



Testbench - fpga_tb.vhd

- Note the use of `tb_utilities_pkg`

```

library work;
use work.tb_utilities_pkg.all;

entity esc_sv09_fpga_tb is
end esc_sv09_fpga_tb;

architecture esc_sv09_fpga_tb_arch of esc_sv09_fpga_tb is
-- Constant declarations
constant RESET_LENGTH : integer := 20;
constant CLK_PERIOD   : time    := 10 ns;
constant Tco : time   := 1 ns; -- Register clock-to-output delay for simulation
constant Tpd : time   := 1 ns; -- Combinatorial logic propagation delay for simulation
-- Component declarations
component esc_sv09_fpga is
generic(
    Tco : time := 1 ns -- Tco for simulation only
);
port(
    pwm_out : out std_logic;
    clk     : in  std_logic;
    reset   : in  std_logic
);
end component esc_sv09_fpga;
-- Signal declarations
signal tb_clk      : std_logic;
signal tb_reset    : std_logic := '1';

```

70



Testbench fpga_tb.vhd

- Generate stimulus
- Instantiate the DUT

```
begin
-- Generate_clock_and_reset_waveforms
tb_reset <= '1','0' after RESET_LENGTH * CLK_PERIOD;

l_clk : generate_clock(
  clk    => tb_clk,
  Tperiod => CLK_PERIOD,
  Tpulse  => CLK_PERIOD/2,
  Tphase  => 0 ns
);

-- Instantiate the device under test (DUT)
DUT : esc_sv09_fpga
generic map(
  Tco => Tco
)
port map(
  pwm_out => open,
  clk     => tb_clk,
  reset   => tb_reset
);
end esc_sv09_fpga_tb_arch;
```

71



Testbench tb_utilities_pkg.vhd

- Can add file read/write routines
- File ideas:
 - “control” file
 - input data from model
 - output data for comparison to model data

```
package tb_utilities_pkg is
  procedure generate_clock(
    signal clk : out std_logic;
    constant Tperiod : in time;
    constant Tpulse : in time;
    constant Tphase : in time
  );
end tb_utilities_pkg;

package body tb_utilities_pkg is
-- A useful routine for generating multiple clocks.
-- from "The Designer's Guide to VHDL", Peter Ashenden

  procedure generate_clock(
    signal clk : out std_logic;
    constant Tperiod : in time;
    constant Tpulse : in time;
    constant Tphase : in time
  )is
  begin
    clk <= '0';
    wait for Tphase;
    loop
      clk <= '1', '0' after Tpulse;
      wait for Tperiod;
    end loop;
  end procedure generate_clock;
end tb_utilities_pkg;
```

72

- Note signal declarations are dependent on data in the Matlab generated VHDL packages

```
library work;
use work.esc_sv09_pwm_pkg.all;
use work.esc_sv09_sin_pkg.all;
```

```
constant FCW : std_logic_vector(Nb_PhAcc-1 downto 0) := X"00_00A8";
constant BIAS : std_logic_vector(7 downto 0) := X"80";
-- Signal declarations
signal pwm : std_logic;
signal pwm_next : std_logic;
signal double_sin : unsigned(TABLE_DATA_WIDTH downto 0);
signal sin : std_logic_vector(TABLE_DATA_WIDTH downto 0);
signal pwm_set_point : std_logic_vector(TABLE_DATA_WIDTH downto 0);
signal pwm_set_point_next : std_logic_vector(TABLE_DATA_WIDTH downto 0);
```

73

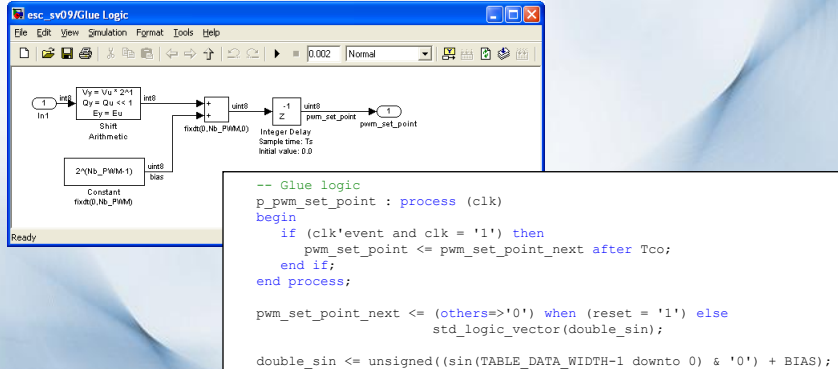
- Component declarations

```
-- Component declarations
component esc_sv09_sin is
generic(
  Tco : time := 1 ns -- Tco for simulation only
);
port(
  fcw_in : in std_logic_vector(Nb_PhAcc-1 downto 0);
  phacc_en_in : in std_logic; -- Phase Accumulator enable
  sin_out : out std_logic_vector(TABLE_DATA_WIDTH downto 0);
  clk : in std_logic;
  reset : in std_logic
);
end component esc_sv09_sin;

component esc_sv09_pwm is
generic(
  Tco : time := 1 ns -- Tco for simulation only
);
port(
  set_point_in : in std_logic_vector(Nb_PWM-1 downto 0);
  pwm_enable_in : in std_logic;
  pwm_out : out std_logic;
  clk : in std_logic;
  reset : in std_logic
);
end component esc_sv09_pwm;
```

74

- There are many VHDL styles
 - Look at coding guidelines from FPGA manufacturer



```

-- Glue logic
p_pwm_set_point : process (clk)
begin
  if (clk'event and clk = '1') then
    pwm_set_point <= pwm_set_point_next after Tco;
  end if;
end process;

pwm_set_point_next <= (others=>'0') when (reset = '1') else
  std_logic_vector(double_sin);

double_sin <= unsigned((sin(TABLE_DATA_WIDTH-1 downto 0) & '0') + BIAS);

```

75

- Component instantiations

```

-- Instantiate components
Inst_esc_sv09_sin : esc_sv09_sin
generic map(
  Tco => Tco
)
port map(
  fcw_in      => FCW,
  phacc_en_in => '1', -- always enabled
  sin_out     => sin,

  clk => clk,
  reset => reset
);

Inst_esc_sv09_pwm : esc_sv09_pwm
generic map(
  Tco => Tco
)
port map(
  set_point_in => pwm_set_point,
  pwm_enable_in => '1', -- always enabled
  pwm_out     => pwm_out,

  clk => clk,
  reset => reset
);

end esc_sv09_fpga_arch;

```

76



Design component sin.vhd

- log2c function does not necessarily add logic...

```

library work;
use work.esc_sv09_sin_pkg.all;

entity esc_sv09_sin is
generic(
    Tco : time := 1 ns -- Tco for simulation only
);
port (
    fcw_in      : in  std_logic_vector(Nb_PhAcc-1 downto 0);
    phacc_en_in : in  std_logic; -- Phase Accumulator enable
    sin_out     : out std_logic_vector(TABLE_DATA_WIDTH downto 0);
    clk        : in  std_logic;
    reset      : in  std_logic
);
end esc_sv09_sin;

architecture esc_sv09_sin_arch of esc_sv09_sin is
-- function to determine the number of bits m required to
-- represent values up to an integer n
function log2c(n : integer) return integer is
    variable m : integer;
    variable p : integer;
begin
    m := 0;
    p := 1;
    while p < n loop
        m := m + 1;
        p := p * 2;
    end loop;
    return m;
end log2c;

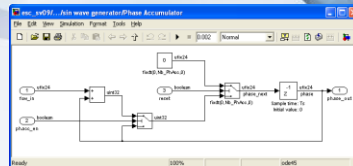
```

77



Design component sin.vhd

- Phase accumulator
- Extract bits (8 MSB's)
- Note use of log2c



```

-- phase accumulator
p_count : process (clk)
begin
    if (clk'event and clk = '1') then
        phase <= phase_next after Tco;
    end if;
end process;

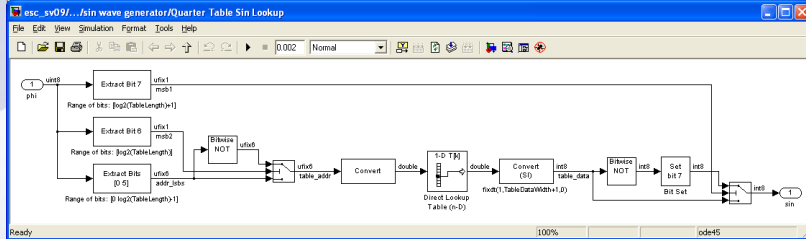
phase_next <= (others=>'0') when (reset = '1') else
    (phase + fcw_in) when (phacc_en_in = '1') else
    phase; -- no change, keep the last value

-- extract bits
-- Use log2c function to avoid re-code for changes in TABLE_LENGTH
phase_bits <= phase((Nb_PhAcc-1) downto (Nb_PhAcc-1)-(log2c(TABLE_LENGTH)+1) );

msb1    <= phase_bits( log2c(TABLE_LENGTH)+1 );
msb2    <= phase_bits( log2c(TABLE_LENGTH) );
addr_lsbs <= phase_bits( log2c(TABLE_LENGTH)-1 downto 0);

```

78



```

-- table lookup
-- Note the register after the memory element. the memory may be a fixed FPGA
-- resource; however, the downstream circuit may be placed arbitrarily.
table_addr <= not(addr_lsbs) when msb2 = '1' else addr_lsbs;

p_table_data : process (clk)
begin
    if (clk'event and clk = '1') then
        table_data <= table_data_next after Tco;
    end if;
end process;

table_data_next <= conv_std_logic_vector(sin_array(conv_integer(table_addr)), table_data'length);
-- form the output
sin_out <= ('1' & not(table_data)) when (msb1 = '1') else ('0' & table_data);

```

79

- Signal declarations are dependent on data in the Matlab generated VHDL packages

```

library work;
use work.esc_sv09_pwm_pkg.all;

entity esc_sv09_pwm is
generic(
    Tco : time := 1 ns -- Tco for simulation only
);
port (
    set_point_in : in std_logic_vector(Nb_PWM-1 downto 0);
    pwm_enable_in : in std_logic;
    pwm_out : out std_logic;

    clk : in std_logic;
    reset : in std_logic
);
end esc_sv09_pwm;

architecture esc_sv09_pwm_arch of esc_sv09_pwm is
-- Signal declarations
signal count : std_logic_vector(Nb_PWM-1 downto 0);
signal count_next : std_logic_vector(Nb_PWM-1 downto 0);
signal pwm : std_logic;
signal pwm_next : std_logic;

```

80



Design component pwm.vhd

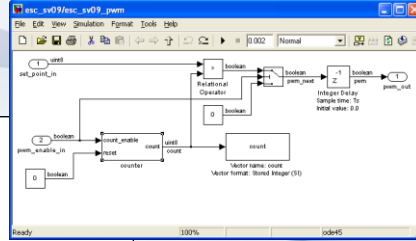
```

-- Free-running counter
p_count : process (clk)
begin
    if (clk'event and clk = '1') then
        if (reset = '1') then
            count <= (others=>'0') after Tco;
        elsif (pwm_enable_in = '1') then
            count <= count_next after Tco;
        end if;
    end if;
end process;

-- PWM Register
p_pwm : process (clk)
begin
    if (clk'event and clk = '1') then
        pwm <= pwm_next after Tco;
    end if;
end process;

pwm_next <= '0' when (reset = '1') or (pwm_enable_in = '0') else
    '1' when (set_point_in > count) else
    '0'; -- always include a default to avoid inferring a latch

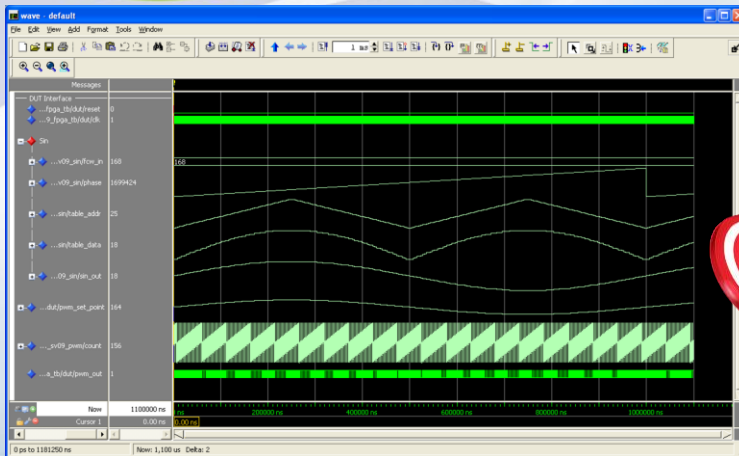
-- assign the output
pwm_out <= pwm;
    
```



81



HDL Simulation

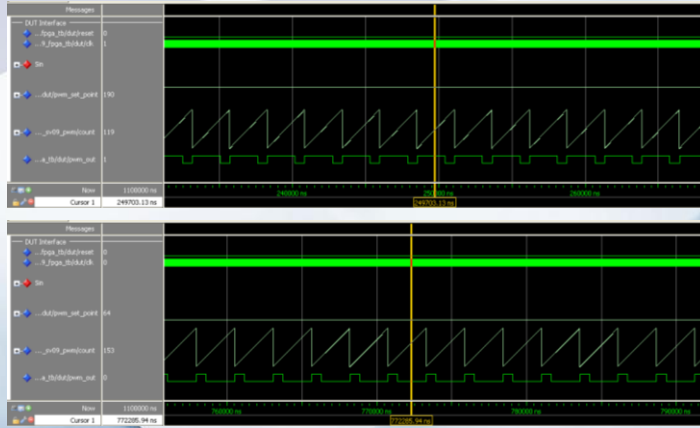


If The FPGA simulation matches the model, the FPGA is assumed to be correct. Explain or correct any value or timing differences.

82



HDL Simulation




Based on the PWM waveform, what is the output of the R-C filter?



Model Based Design for FPGA Development


Conclusions



Conclusions from HDL Simulation

- The functionality of the digital design as embodied in the HDL matches the implementation model.
 - The HDL simulation verifies that we made the thing right
- The HDL simulation cannot validate the digital design
 - It cannot prove that we made the right thing
- With only the HDL simulation, you do not know the system behavior


85



Conclusions from Model Based Design

- The implementation model gives a very high degree of confidence that if the FPGA matches the model, the FPGA will meet system requirements
 - If the FPGA matches the model...
 - ...and the model satisfies the system requirements
- Then the FPGA will do the right thing in the system


86



Further Study

- Courses
 - Xilinx “DSP for FPGA Primer”
 - Altera
- Webinars
 - The Mathworks “Creating FPGA-based Coprocessors for DSP Using Model-Based Design”
- Books
 - “FPGA Based Implementation of Signal Processing Systems”, Woods et al., Wiley 2008
- Websites
 - www.andraka.com - DSP with FPGAs

87



Wrap-up & Questions

- Example files are available at: www.irtc-hq.com

88