Accelerate FPGA Prototyping with MATLAB and Simulink

September 21st 2010

Stephan van Beek
Senior Application Engineer
Need for FPGA prototyping:
- Real-Time Simulation
- Modeling accuracy
- Interfacing with peripherals
Key Takeaways

- Automation of manual steps in FPGA prototyping allowing shorter iteration cycles
- Integration of FPGA development tools enhances verification
- Automatic HDL Code generation can be adapted to meet your requirement
From Idea to Implementation (manual)

- Fixed Point Conversion
- HDL Code Creation
- HDL Verification
- FPGA Prototyping
MathWorks Solutions

- Algorithm Development
- MATLAB Simulink Stateflow
- Fixed-Point

IMPLEMENTATION
- C, C++
- VHDL, Verilog
- SPICE
  - MCU
  - DSP
  - FPGA
  - ASIC
  - Analog Hardware

TEST & VERIFICATION
- Fixed Point Conversion
- HDL Code Creation
- HDL Verification
- FPGA Prototyping
MathWorks Solutions

DESIGN
- Algorithm Development
- MATLAB Simulink Stateflow
- Fixed-Point

IMPLEMENTATION
- C, C++
- VHDL, Verilog
- SPICE
- MCU
- DSP
- FPGA
- ASIC
- Analog Hardware

TEST & VERIFICATION
- Fixed Point Conversion
- HDL Code Creation
- HDL Verification
- FPGA Prototyping

INTEGRATION
How to Handle Fixed-Point Conversion?

MATLAB

Simulink

Stateflow

FPGA

Pi = 3.141592653589793…

Pi = 3.141601562500000
Signed, 14-bits, 11 bits fraction
- Resources?
- Overflow?
- Precision? Error=8.9089e-6

Pi = 01100100100010

MathWorks solution:
Fixed-Point Tools
Simulink Fixed-Point

- Convert floating point to fixed point models
  - Automatic tracking of signal range (also intermediate quantities)
  - Fraction lengths recommendation
- Bit-true models in the same environment
MathWorks Solutions

- Fixed Point Conversion
- **HDL Code Creation**
- HDL Verification
- FPGA Prototyping
BEGIN
-- Block Statements
ce_output : PROCESS (clk, reset)
BEGIN
IF reset = '1' THEN
  ring_count <= to_unsigned(1, 3);
ELSIF clk'event AND clk = '1' THEN
  IF enb_1_1_1 = '1' THEN
    ring_count <= ring_count(0) & ring_count(2 DOWNTO 1);
  END IF;
END IF;
END PROCESS ce_output;

USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.ALL;

ENTITY Stage1 IS
  PORT( clk : IN std_logic;
        enb_1_1_1                       : IN std_logic;
        reset                           : IN std_logic;
        Stage1_in_re                        : IN std_logic_vector(17 DOWNTO 0);
        Stage1_in_im                        : IN std_logic_vector(35 DOWNTO 0);
        Stage1_out_re                       : OUT std_logic;
        Stage1_out_im                       : OUT std_logic_vector(35 DOWNTO 0);
      );
END Stage1;

ARCHITECTURE rtl OF Stage1 IS

-- Signals
SIGNAL ring_count : unsigned(2 DOWNTO 0); -- ufix3
SIGNAL phase_0                          : std_logic; -- boolean
SIGNAL phase_1                          : std_logic; -- boolean
SIGNAL phase_2                          : std_logic; -- boolean
SIGNAL input_typeconvert_re             : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL input_typeconvert_im             : signed(35 DOWNTO 0); -- sfix36_En34
SIGNAL product_phase0_2_re              : signed(47 DOWNTO 0); -- sfix48_En48
SIGNAL product_phase0_2_im              : signed(47 DOWNTO 0); -- sfix48_En48
SIGNAL mul_temp_2                       : signed(35 DOWNTO 0); -- sfix36_En34
SIGNAL mul_temp_3                       : signed(35 DOWNTO 0); -- sfix36_En34
SIGNAL product_phase1_1_re              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase1_2_im              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase1_2_re              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase1_1_im              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase2_1_im              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase2_2_im              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase2_2_re              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase2_1_re              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase2_0_re              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase2_0_im              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase1_0_re              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase1_0_im              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase0_1_re              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase0_1_im              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase0_0_re              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL product_phase0_0_im              : signed(17 DOWNTO 0); -- sfix18_En16
SIGNAL mul temp_1                       : signed(35 DOWNTO 0); -- sfix36_En34
SIGNAL mul temp_2                       : signed(35 DOWNTO 0); -- sfix36_En34
SIGNAL mul temp_3                       : signed(35 DOWNTO 0); -- sfix36_En34
SIGNAL mul temp_4                       : signed(35 DOWNTO 0); -- sfix36_En34
SIGNAL mul temp_5                       : signed(35 DOWNTO 0); -- sfix36_En34
SIGNAL mul temp_6                       : signed(35 DOWNTO 0); -- sfix36_En34

BEGIN
-- Module Architecture: Stage1
ARCHITECTURE rtl OF Stage1 IS

-- Local Functions
-- Type Definitions
TYPE input_pipeline_type IS ARRAY (NATURAL range <>) OF signed(17 DOWNTO 0); -- sfix18_En16

-- Constants
CONSTANT coeffphase1_1 : signed(17 DOWNTO 0) := to_signed(10991, 18); -- sfix18_En18
CONSTANT coeffphase1_2 : signed(17 DOWNTO 0) := to_signed(63566, 18); -- sfix18_En18
CONSTANT coeffphase2_1 : signed(17 DOWNTO 0) := to_signed(20991, 18); -- sfix18_En18
CONSTANT coeffphase2_2 : signed(17 DOWNTO 0) := to_signed(89431, 18); -- sfix18_En18

-- Signals
SIGNAL ring_count : unsigned(2 DOWNTO 0); -- ufix3
SIGNAL phase_0 : std_logic; -- boolean
SIGNAL phase_1 : std_logic; -- boolean
SIGNAL phase_2 : std_logic; -- boolean
SIGNAL input_typeconvert_re : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL input_typeconvert_im : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase0_2_re : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase0_2_im : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase1_2_re : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase1_2_im : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase1_1_re : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase1_1_im : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase2_1_re : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase2_1_im : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase2_2_re : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase2_2_im : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase2_0_re : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase2_0_im : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase1_0_re : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase1_0_im : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase0_1_re : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase0_1_im : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase0_0_re : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL product_phase0_0_im : signed(17 DOWNTO 0); -- sfix18_En18
SIGNAL mul temp_1 : signed(35 DOWNTO 0); -- sfix36_En34
SIGNAL mul temp_2 : signed(35 DOWNTO 0); -- sfix36_En34
SIGNAL mul temp_3 : signed(35 DOWNTO 0); -- sfix36_En34
SIGNAL mul temp_4 : signed(35 DOWNTO 0); -- sfix36_En34
SIGNAL mul temp_5 : signed(35 DOWNTO 0); -- sfix36_En34
SIGNAL mul temp_6 : signed(35 DOWNTO 0); -- sfix36_En34

BEGIN
IF reset = '1' THEN
  ring_count <= to_unsigned(1, 3);
ELSIF clk'event AND clk = '1' THEN
  IF enb_1_1_1 = '1' THEN
    ring_count <= ring_count(0) & ring_count(2 DOWNTO 1);
  END IF;
END IF;
END PROCESS ce_output;

phase_0 <= ring_count(0) AND enb_1_1_1;
phase_1 <= ring_count(1) AND enb_1_1_1;
phase_2 <= ring_count(2) AND enb_1_1_1;
input_typeconvert_re <= signed(Stage1_in_re);
input_typeconvert_im <= signed(Stage1_in_im);

Delay_Pipeline_Phase0_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase0_re <= (OTHERS => '0');
    input_pipeline_phase0_im <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      input_pipeline_phase0_re <= input_typeconvert_re;
      input_pipeline_phase0_im <= input_typeconvert_im;
    END IF;
  END IF;
END PROCESS Delay_Pipeline_Phase0_process;

Delay_Pipeline_Phase1_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase1_re(0 TO 1) <= (OTHERS => '0');
    input_pipeline_phase1_im(0 TO 1) <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF phase_0 = '1' THEN
      input_pipeline_phase0_re <= input_typeconvert_re;
      input_pipeline_phase0_im <= input_typeconvert_im;
    END IF;
  END IF;
END PROCESS Delay_Pipeline_Phase0_process;

Delay_Pipeline_Phase1_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase1_re(0 TO 1) <= (OTHERS => '0');
    input_pipeline_phase1_im(0 TO 1) <= (OTHERS => '0');
  ELSIF clk'event AND clk = '1' THEN
    IF phase_1 = '1' THEN
      input_pipeline_phase1_re <= input_pipeline_phase0_re;
      input_pipeline_phase1_im <= input_pipeline_phase0_im;
    END IF;
  END IF;
END PROCESS Delay_Pipeline_Phase0_process;

Delay_Pipeline_Phase2_process : PROCESS (clk, reset)
BEGIN
  IF reset = '1' THEN
    input_pipeline_phase2_re <= (OTHERS => '0');
  END IF;
END PROCESS Delay_Pipeline_Phase0_process;

END Stage1;
END rtl;
Manual HDL Code Creation

- Typical HDL designs contain many lines of code
- Days or maybe weeks to develop?
- How to implement Fixed Point in HDL?
- What if the specification changes?

MathWorks solution: Automatic HDL Code Generation
Simulink HDL Coder generates bit-true, cycle-accurate, synthesizable HDL code from Simulink models, Stateflow charts, and MATLAB code.

Accelerates generation and verification of vendor independent, readable RTL code.
MathWorks Solutions

DESIGN
- Algorithm Development
- MATLAB Simulink Stateflow
- Fixed-Point

IMPLEMENTATION
- C, C++
- VHDL, Verilog
- SPICE
  - MCU
  - DSP
  - FPGA
  - ASIC
  - Analog Hardware

TEST & VERIFICATION
- Fixed Point Conversion
- HDL Code Creation
- HDL Verification
- FPGA Prototyping

INTEGRATION
HDL Verification

- Design the Test Bench twice
- Many stimuli-files from MATLAB
- These are ideal references which require pre- and post-processing
- How to analyze results?

MathWorks solution:
Re-Use System Level Testbench
Co-Simulation with HDL Simulator

Simulink

Test Bench

Stimuli

Algorithm

EDA Simulator Link

Results

Test Bench

HDL Simulator

Difference
Co-Simulation with HDL Simulator

- Re-use of MATLAB/Simulink testbench
- Extended analysis capabilities
- Dynamic Testbench (closed loop verification, multi domain)
- Automatic creation of co-simulation models
- Integrating handwritten HDL code
MathWorks Solutions

- Fixed Point Conversion
- HDL Code Creation
- HDL Verification
- **FPGA Prototyping**
FPGA Prototyping

- Building confidence
- Tool flow automation
- Interfacing with peripherals

**MathWorks solution:** Integration with 3rd party solutions
FPGA-in-the-Loop Simulation

Simulink

Test Bench → Stimuli → Algorithm → Results → Test Bench

FPGA Target

FPGA development board
Automated workflow ➔ from model to FPGA prototyping
HDL Coder Only for Prototyping?

Folding to reduce area

Automatically add Pipelining to improve timing

Clk = 10MHz

Clk = 60MHz
Summary

- Automation of manual steps in FPGA prototyping allowing shorter iteration cycles
  - Assisted Fixed Point Conversion
  - Automatic HDL Code Generation
  - FPGA Turnkey Flow

- Integration of FPGA development tools enhances verification
  - Improved analysis, closed loop verification, multi domain

- Automatic HDL Code generation can be adapted to meet your requirements
  - Optimization for area and speed
Next steps ..... 

- Visit [www.mathworks.nl/fpga](http://www.mathworks.nl/fpga) for more information

- Request a **free ‘guided’ Simulink HDL Coder trial**
  - Contact your local sales rep

- **Watch our Simulink HDL Coder webinars:**
  - [http://www.mathworks.nl/company/events/webinars](http://www.mathworks.nl/company/events/webinars)

**Questions??**