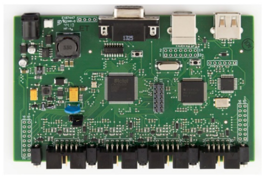
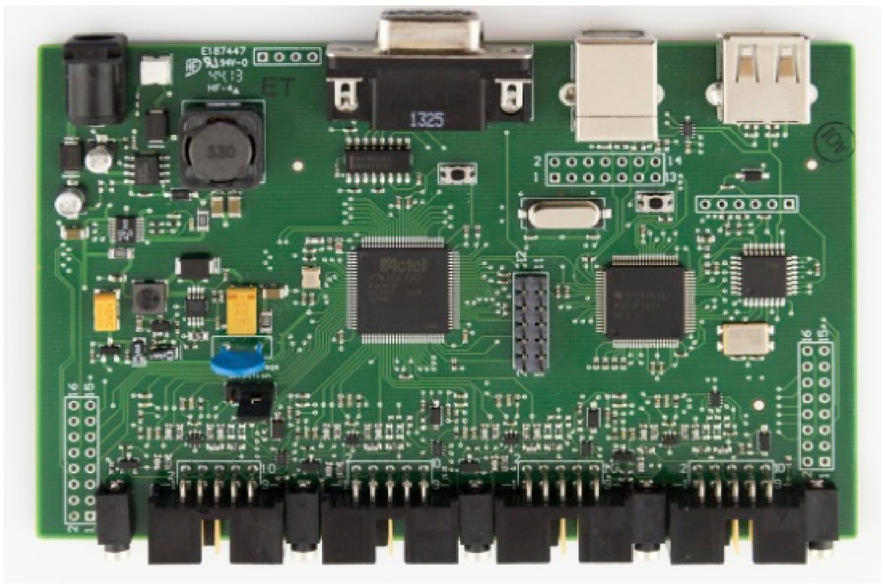


Actel SmartDesign MSS ACE Simulation User Guide

Contents

Actel SmartDesign MSS ACE Simulation



Product Information: SmartDesign MSS ACE Simulation

The SmartDesign MSS ACE Simulation is a feature that allows simulation of the ACE functionality in ModelSimTM. It provides users with a way to verify that their configuration works based on their system input. This tool is a part of the SmartFusion MSS, and it includes a library of analog drivers functions. The tool provides a simple example of simulating ACE, and the user manual provides detailed instructions for users to get started.

Creating the Design

To create a design using the SmartDesign MSS ACE Simulation tool, users will need to follow these steps:

1. Configure MSS: Disable peripherals that are not needed and create a simple ACE configuration.
2. Create a top-level SmartDesign wrapper and instantiate the configured MSS component.
3. Prepare the testbench: Customize the basic testbench to include ACE simulations.

Configuring MSS

In this step, users will need to disable the peripherals that are not needed for their specific configuration. The following peripherals can be disabled:

- UARTs
- SPIs
- I2Cs
- MAC
- Fabric Interface
- External Memory Controller

After disabling unnecessary peripherals, users can create a simple ACE configuration consisting of a single ADC Direct Input service with a few flags and a sampling sequence loop. The Flag mapping feature can be used to determine which Flag register and bits flags were mapped to. Once the configuration is complete, the MSS design can be generated.

Preparing the Testbench

The final step is to prepare the testbench. SmartDesign automatically generates a testbench.v file that is useful for basic simulations. However, users will need to customize this file to include ACE simulations. Users can create a custom testbench by following these steps:

1. Open the Files tab in the Project Manager to view the file hierarchy.
2. Locate the testbench.v file and customize it to include ACE simulations.

Once the testbench is customized, users can simulate the ACE functionality in ModelSimTM.

SmartDesign MSS
ACE Simulation

Introduction

The ACE functionality can be simulated in ModelSim™ to verify that your configuration works based on your system input. This document walks through a simple example of simulating the ACE. Please refer to Simulating the Microcontroller Subsystem for a more general overview of the simulation strategy for SmartFusion MSS. Details about the analog driver functions that are available in the SmartFusion library are at the end of this document in the Analog Drivers section.

Creating the Design

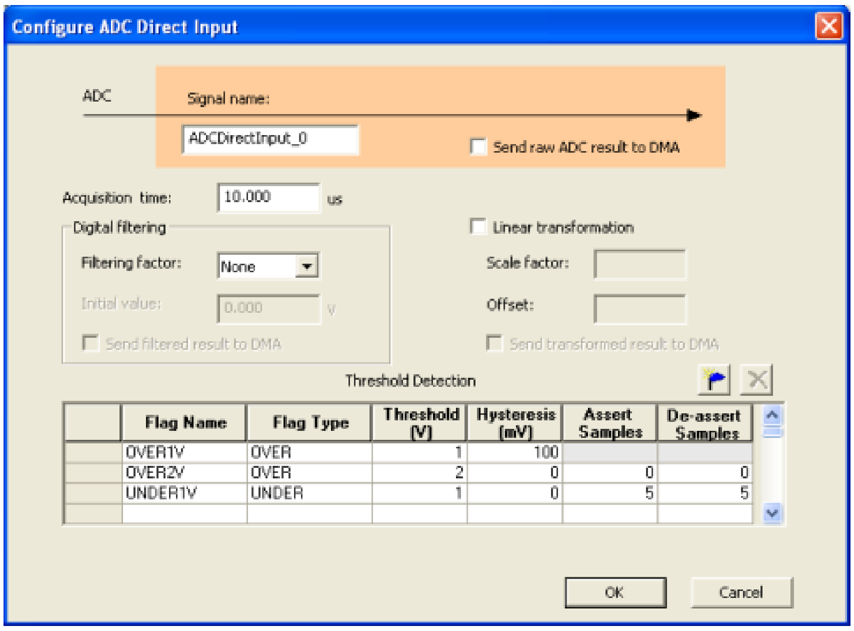
We will create a simple SmartFusion MSS and ACE configuration to demonstrate how you can simulate the ACE.

Configuring MSS

We'll disable the following peripherals since we will not be using them in this example:

- UARTs
- SPIs
- I2Cs
- MAC
- Fabric Interface
- External Memory Controller

We'll create a simple ACE configuration consisting of a single "ADC Direct Input" service with a few flags, and a simple sampling sequence loop. The configuration is shown below.



Signal name: ADCDirectInput_0

Send raw ADC result to DMA

Acquisition time: 10.000 us

Digital filtering

Filtering factor: None

Initial value: 0.000 V

Send filtered result to DMA

Linear transformation

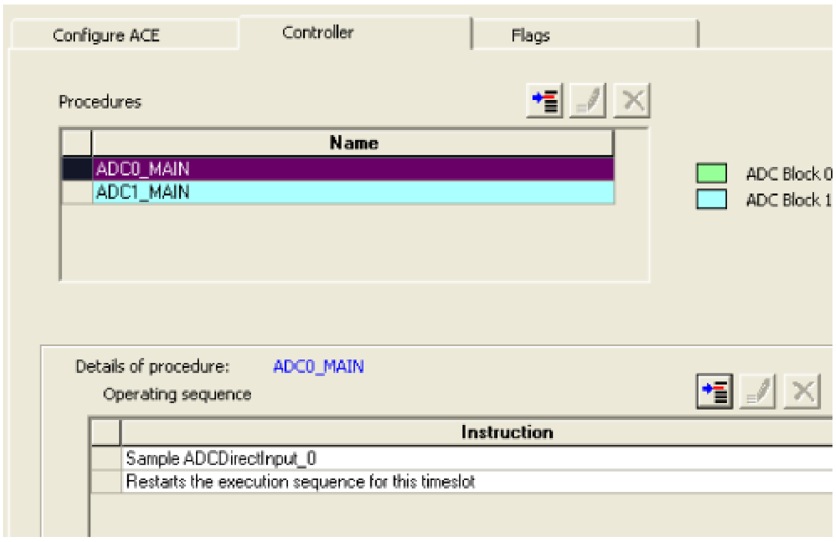
Scale factor:

Offset:

Send transformed result to DMA

Flag Name	Flag Type	Threshold (V)	Hysteresis (mV)	Assert Samples	De-assert Samples
OVER1V	OVER	1	100		
OVER2V	OVER	2	0	0	0
UNDER1V	UNDER	1	0	5	5

Figure 1: ADC Direct Input and Threshold Configuration



Procedures

Name
ADC0_MAIN
ADC1_MAIN

🟩 ADC Block 0
🟦 ADC Block 1

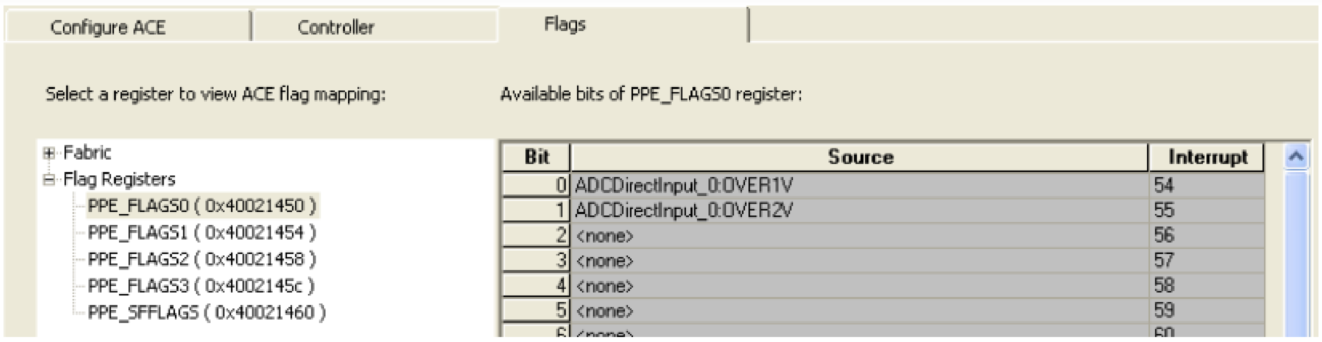
Details of procedure: ADC0_MAIN

Operating sequence

Instruction
Sample ADCDirectInput_0
Restarts the execution sequence for this timeslot

Figure 2: Sampling Sequence Configuration

We use the Flags tab to determine which Flag register and bits our flags were mapped to. This is useful when we write our BFM script later (as shown in the figure below).

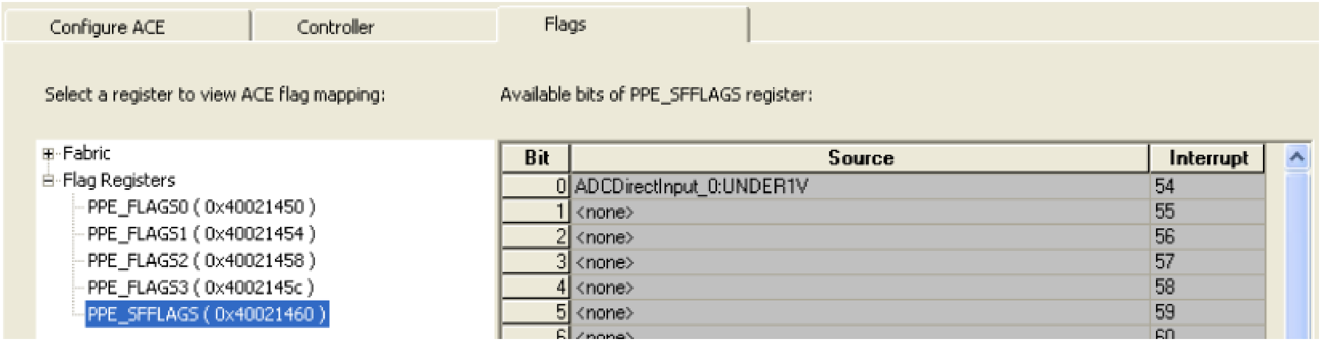


Select a register to view ACE flag mapping:

- Fabric
- Flag Registers
  - PPE_FLAGS0 (0x40021450)
  - PPE_FLAGS1 (0x40021454)
  - PPE_FLAGS2 (0x40021458)
  - PPE_FLAGS3 (0x4002145c)
  - PPE_SFFLAGS (0x40021460)

Available bits of PPE_FLAGS0 register:

Bit	Source	Interrupt
0	ADCDirectInput_0:OVER1V	54
1	ADCDirectInput_0:OVER2V	55
2	<none>	56
3	<none>	57
4	<none>	58
5	<none>	59



Select a register to view ACE flag mapping:

- Fabric
- Flag Registers
  - PPE_FLAGS0 (0x40021450)
  - PPE_FLAGS1 (0x40021454)
  - PPE_FLAGS2 (0x40021458)
  - PPE_FLAGS3 (0x4002145c)
  - PPE_SFFLAGS (0x40021460)

Available bits of PPE_SFFLAGS register:

Bit	Source	Interrupt
0	ADCDirectInput_0:UNDER1V	54
1	<none>	55
2	<none>	56
3	<none>	57
4	<none>	58
5	<none>	59

Figure 3: Flag Mapping

The Flag mapping tells us that

- OVER1V was mapped to PPE_FLAGS0 register, bit 0
- OVER2V was mapped to PPE_FLAGS0 register, bit 1
- UNDER1V was mapped to PPE_SFFLAGS0 register, bit 0

Our MSS design should look like this after configuration:

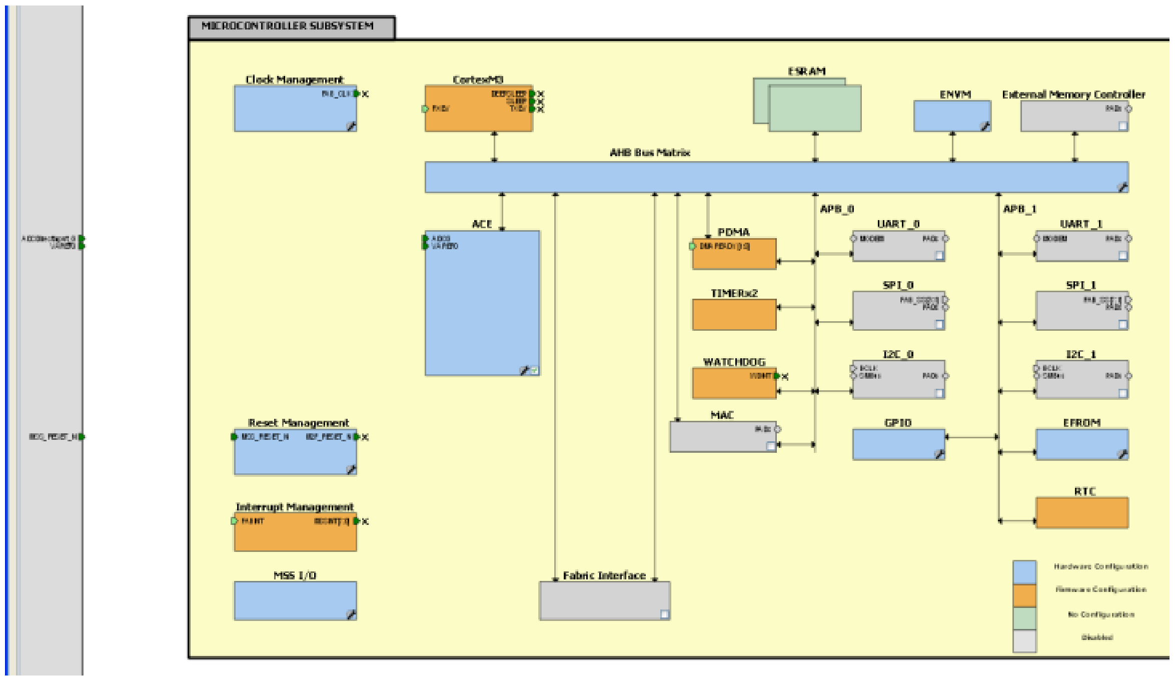


Figure 4: Sample MSS Design After Configuration

Generate the MSS

Creating Top Level SmartDesign Wrapper

Create a top level SmartDesign component and instantiate our newly configured MSS component. Set the top level SmartDesign as root, and generate the SmartDesign.

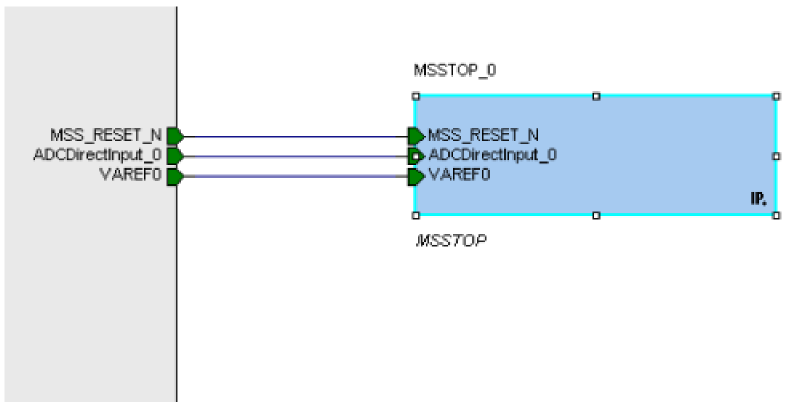


Figure 5: Top-Level SmartDesign on the Canvas

Preparing the Testbench

Now that the design is generated, let's open up two files that we'll need for simulation purposes.
Go to the Libero® IDE Project Manager Files tab and open the testbench.v and user.bfm from your MSS component (as shown in the figure below).

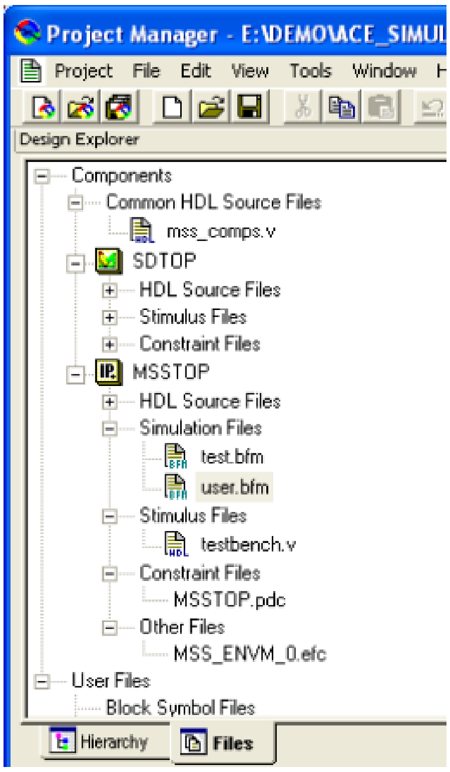


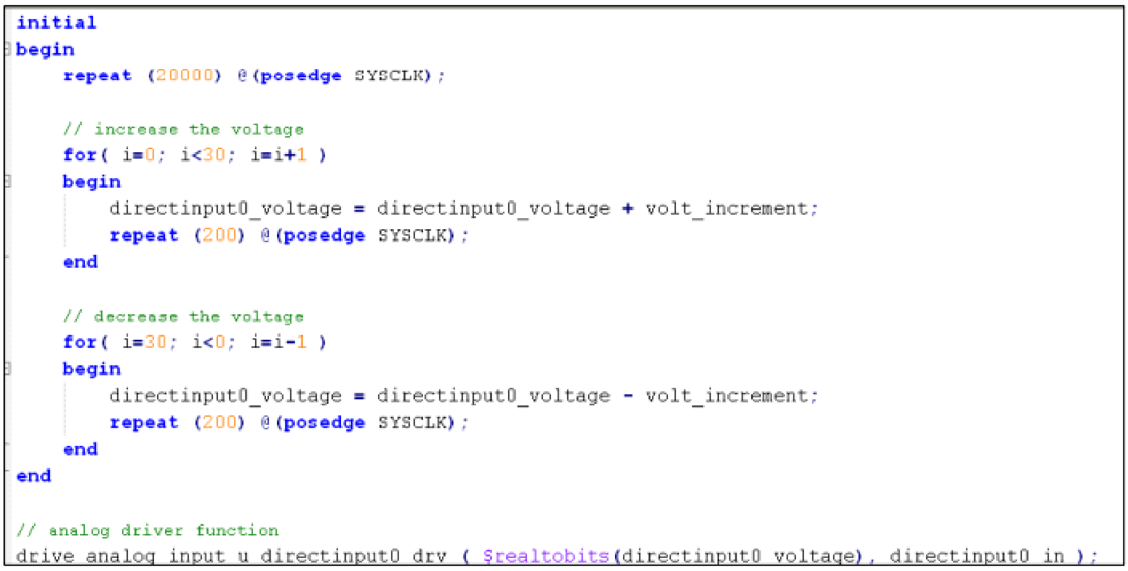
Figure 6: Files Tab (File Hierarchy) in Project Manager

Creating a Custom Testbench

The testbench.v file that is automatically generated by SmartDesign is useful for basic simulations, but for ACE simulations we will need to customize this basic testbench.

To create a new testbench:

1. From the Libero IDE Project Manager choose File > New.
   a. Select HDL Stimulus File
   b. Name the file ace_testbench and click OK.
2. Copy and paste the contents of testbench.v to ace_testbench. We now have a testbench that we can customize for ACE simulations.
3. Add a simple SmartFusion CAE library analog driver function to drive our analog input service ADCDirectInput. The following code fragment should be added to your testbench. A voltage value is ramped up, then down.



```
initial
begin
    repeat (20000) @(posedge SYSCLK);

    // increase the voltage
    for( i=0; i<30; i=i+1 )
    begin
        directinput0_voltage = directinput0_voltage + volt_increment;
        repeat (200) @(posedge SYSCLK);
    end

    // decrease the voltage
    for( i=30; i<0; i=i-1 )
    begin
        directinput0_voltage = directinput0_voltage - volt_increment;
        repeat (200) @(posedge SYSCLK);
    end
end

// analog driver function
drive_analog_input_u directinput0_drv ( $realtobits(directinput0_voltage), directinput0_in );
```

Notice the drive_analog_input function that is used to convert the real value into a value that can be driven into the analog port. Refer to the CAE Analog Drivers section for more details.

Modifying our BFM Script

We will create a simple BFM script that just loops and reads our PPE registers. This mimics a Cortex M3 polling scheme. The addresses of the PPE_FLAGSn and PPE_SFFLAGS registers are available in the SmartFusion Handbook. It is also shown in the Flags tab in the ACE configurator, in addition to the bit in which the flag is assigned to in the register.

In the user.bfm script file, we will add these commands:

```
# ACE register offsets
constant PPE_FLAGS0  0x1450;
constant PPE_SFFLAGS 0x1460;

procedure user_main;

# uncomment the following include if you have soft peripherals in the fabric
# that you want to simulate.  The subsystem.bfm file contains the memory map
# of the soft peripherals.
# include "subsystem.bfm"

# add your BFM commands below:
int flags0_value;
int sflag_value;

int loop;
set loop 1;
while loop == 1
  readstore w ACE  PPE_FLAGS0  flags0_value;
  readstore w ACE  PPE_SFFLAGS sflag_value;
endwhile

return
```

In this script, we continually read the PPE_FLAGS0 and PPE_SFFLAGS register addresses into 2 data variables. If we wanted to create a more complex scenario, we could take those values and write them to GPIOs or perform other actions in our BFM commands based upon their value.

Associating our Custom Testbench with our Design

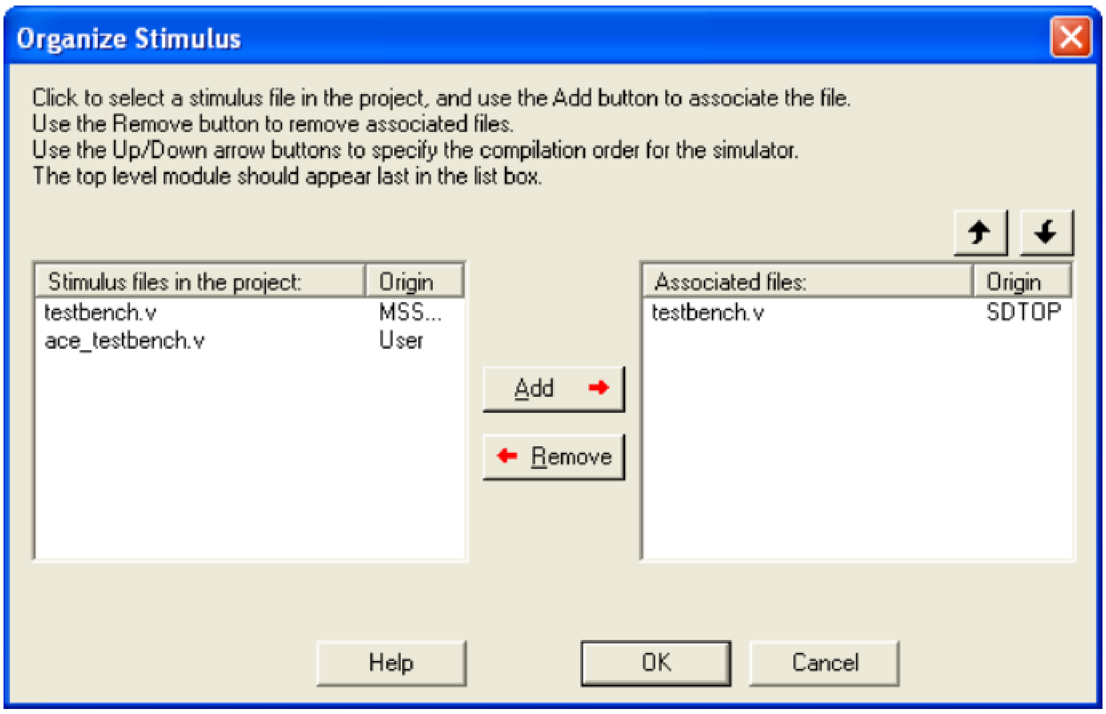
We need to tell the Libero IDE to use our custom testbench for simulation instead of the system generated one.

1. Right-click the SDTOP component in the Project Manager Design Hierarchy and choose Organize Stimulus
2. We want to use ace_testbench instead of testbench.v. So select testbench.v from the right panel and click Remove. Then select ace_testbench.v from the left panel and click Add.
3. Click OK

Simulate

Now we're ready to simulate.

- In the Project Manager Project Flow window click the ModelSim button.
- In ModelSim's command window type run 3ms. In our example, we are running for 3ms because we have a long hardcoded delay in our testbench, because we want to ensure that the ADC calibration is completed before we begin processing.


Organize Stimulus

Click to select a stimulus file in the project, and use the Add button to associate the file.
Use the Remove button to remove associated files.
Use the Up/Down arrow buttons to specify the compilation order for the simulator.
The top level module should appear last in the list box.

Stimulus files in the project:	Origin
testbench.v	MSS...
ace_testbench.v	User

Associated files:	Origin
testbench.v	SDTOP

CAE Analog Drivers

- Analog ports are represented by a 1-bit wide port in both the Verilog and VHDL simulation models. Driver modules are developed to drive a real value through a 1-bit port and to read an analog value from a 1-bit port.
- The drive module/function serializes and streams the real value represented in floating point representation (64-bit value) in zero simulation time, using delta delays. The read module deserializes a stream into a 64-bit value.

Interfaces of all the drivers are given later in respective testbenches

- drive_analog_io and drive_analog_input can drive an analog input. Input is provided to this module as 64 bit value.
- read_analog_io can read any analog signal coming from the Analog Block. Output is provided as a 64 bit value.
- drive_temperature_monitor is used to drive the temperature pad. This module takes temperature in Celsius and converts it into a voltage and drives it over the digital input.
- drive_current_monitor or drive_current_inputs can be used to drive the current pad that will be used for Current Monitoring. As an input it takes the voltage at AT pad, the resistor and current values, to calculate the voltage on the AC quad.
  - Equation is AC(V) = AT(V) + Resistor * current
  - Interface information of both the drivers is given below

Connecting Analog Ports with Verilog

Use $realtobits function to convert the real value to 64 bit value or $bitstoreal function can be used to convert the data from 64 bit to real value.
The following shows the analog drivers that are available in Verilog:

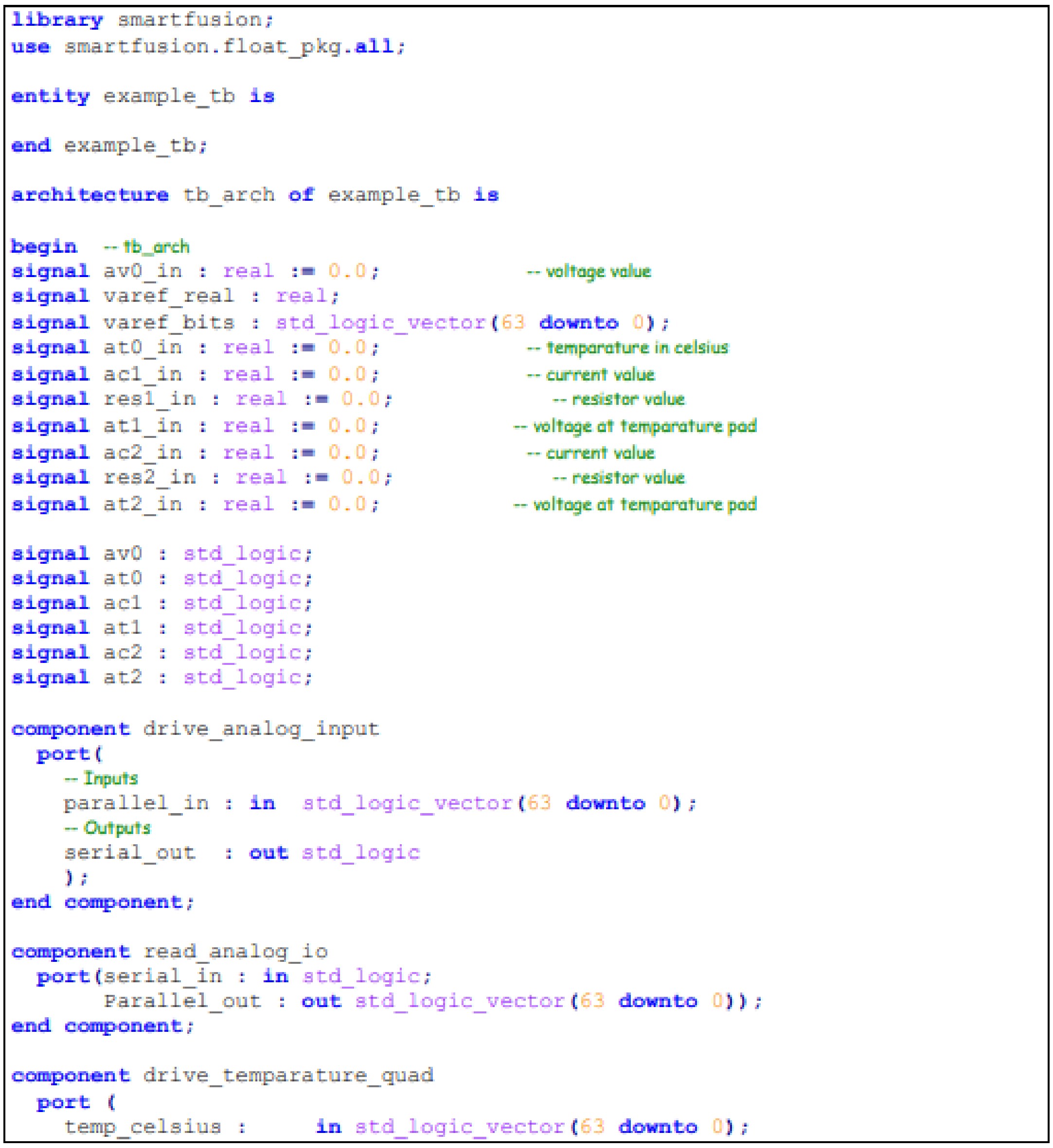
```
module drive_analog_io ( parallel_in, serial_out );
  input  [63:0]  parallel_in;
  output         serial_out;
endmodule

module drive_analog_input ( parallel_in, serial_out );
  input  [63:0]  parallel_in;
  output         serial_out;
endmodule

module drive_current_monitor ( temp_vect, resistor_vect, current_vect,
serial_out );
  input  [63:0]  temp_vect;
  input  [63:0]  resistor_vect;
  input  [63:0]  current_vect;
  output         serial_out;
endmodule

module drive_current_inputs (current_vect, resistor_vect, temp_vect, ac, at );
  input  [63:0]  temp_vect;
  input  [63:0]  resistor_vect;
  input  [63:0]  current_vect;
  output         ac;
  output         at;
endmodule

module drive_temperature_quad ( temp_celsius, serial_out );
  input  [63:0]  temp_celsius;
  output         serial_out;
endmodule

module read_analog_io ( serial_in, read_enb, parallel_out );
  input          serial_in;
  input          read_enb;
  output reg [63:0]  parallel_out;
endmodule
```

The following testbench demonstrates the usage of all drivers.

```
module example_tb ();

real varef real;
real av0 in = 1.0;
real at0 in = 20.0;
real ac1 in = 1.0;
real res1_in = 0.1;
real at1_in = 0.5;
real ac2_in = 1.0;
real res2_in = 0.1;
real at2_in = 0.5;
wire av0, at0, ac1, at1, ac2, at2;

wire [63:0] varef_bits;

//drive voltage input
drive_analog_input inst0 ($realtobits(av0_in), av0);

//Read analog output
read_analog_input inst1(varefout, varef_bits);
always @(varef_bits)
    varef_real = $bitstoreal(varef_bits);

//Drive temperature quad where at0 in is in 0C
drive_temperature_quad inst2($realtobits(at0_in), at0);

//Drive current monitor. ac1_in is current in A. res1_in is resistance value
//in ohms and at1 in is voltage at at1 pad.
drive_current_monitor inst3 ($realtobits(at1_in), $realtobits(res1_in),
$realtobits(ac1_in), ac1);
drive_analog_input inst0 ($realtobits(at1_in), at1);

//Drive current inputs. ac2 in is current in A. res2_in is resistance value
//in ohms and at2 in is voltage at at1 pad.
drive_current_inputs inst4 ($realtobits(ac2_in), $realtobits(res2_in),
$realtobits(at2_in), ac2, at2);
endmodule
```

Connecting Analog Ports with VHDL

realtobits function (equivalent to $realtobits system task in verilog) and bitstoreal function (equivalent to $bitstoreal in verilog) are available in float_pkg package present in smartfusion library. Notice that this package is added to the testbench at the beginning. realtobits can be used to convert the real value to 64 bit floating point representation. bitstoreal function is available in float_pkg package to convert this 64 bit value to a real value. The following testbench demonstrates the usage of all drivers.



```
library smartfusion;
use smartfusion.float_pkg.all;

entity example_tb is

end example_tb;

architecture tb_arch of example_tb is

begin   -- tb_arch
signal av0_in : real := 0.0;              -- voltage value
signal varef_real : real;
signal varef_bits : std_logic_vector(63 downto 0);
signal at0_in : real := 0.0;              -- temparature in celsius
signal ac1_in : real := 0.0;              -- current value
signal res1_in : real := 0.0;               -- resistor value
signal at1_in : real := 0.0;             -- voltage at temparature pad
signal ac2_in : real := 0.0;              -- current value
signal res2_in : real := 0.0;               -- resistor value
signal at2_in : real := 0.0;             -- voltage at temparature pad

signal av0 : std_logic;
signal at0 : std_logic;
signal ac1 : std_logic;
signal at1 : std_logic;
signal ac2 : std_logic;
signal at2 : std_logic;

component drive_analog_input
  port(
    -- Inputs
    parallel_in : in  std_logic_vector(63 downto 0);
    -- Outputs
    serial_out  : out std_logic
    );
end component;

component read_analog_io
  port(serial_in : in std_logic;
       Parallel_out : out std_logic_vector(63 downto 0));
end component;

component drive_temparature_quad
  port (
    temp_celsius :     in std_logic_vector(63 downto 0);
    serial out    : out std_logic);
end component;

component drive_current_monitor
  port (
    temp vect   : in  std logic vector(63 downto 0);
    resistor vect   : in  std logic vector(63 downto 0);
    current vect   : in  std logic_vector(63 downto 0);
    serial_out : out std_logic);
end component;

component drive_current_inputs
  port (
    current vect   : in  std logic vector(63 downto 0);
    resistor vect   : in  std logic vector(63 downto 0);
    temp vect   : in  std_logic_vector(63 downto 0);
    ac : out std logic;
    at : out std_logic);
end component;

begin

--drive voltage input
u_drv_av0 : drive_analog_input
port map (parallel_in => realtobits(av0_in),
               serial_out => av0);

--Read analog output
u read varef : read_analog_ip
  port map (
    serial in     => varefout,
    parallel_out => varef_bits);

varef_real <= bitstoreal(varef_bits);

-- Drive temperature quad where at0_in is in 0C
u_drv_at0 : drive_temparature_quad
  port map (
    temp_celsius => realtobits(at0_in),
    serial_out    => at0);

--Drive current monitor. ac1 in is current in A. res1_in is resistance value,
--in ohms and at1 in is voltage at at1 pad.
u drv ac1 : drive_current_monitor
  port map (
    temp vect => realtobits(at1 in),
    res vect => realtobits(res1 in),
    current vect => realtobits(ac1_in),
    serial out    => ac1);
u drv at1 : drive analog input
port map (parallel in => realtobits(at1_in),
               serial_out => at1);

--Drive current inputs. ac2 in is current in A. res2_in is resistance value --
in ohms and at2_in is voltage at "at"
```

```
u drv ac2 : drive_current_inputs
  port map (
    temp_vect => realtobits(at2_in),
    res_vect => realtobits(res2_in),
    current_vect => realtobits(ac2_in),
    ac    => ac2,
    at    => at2);

end tb_arch;
```

Actel is the leader in low-power and mixed-signal FPGAs and offers the most comprehensive portfolio of system and power management solutions. Power Matters. Learn more at 14H http://www.actel.com

Actel Corporation
2061 Stierlin Court Mountain View, CA 94043-4655 USA Phone 650.318.4200 Fax 650.318.4600
Actel Europe Ltd.
River Court, Meadows Business Park Station Approach, Blackwater Camberley Surrey GU17 9AB United Kingdom
Phone +44 (0) 1276 609 300 Fax +44 (0) 1276 607 540

Actel Japan
EXOS Ebisu Building 4F 1-24-14 Ebisu Shibuya-ku Tokyo 150, Japan Phone +81.03.3445.7671 Fax +81.03.3445.7668 15H **http://jp.actel.com**
Actel Hong Kong
Room 2107, China Resources Building 26 Harbour Road
Wanchai, Hong Kong
Phone +852 2185 6460
Fax +852 2185 6488
www.actel.com.cn

Documents / Resources

SmartDesign MSS ACE Simulation Doc Version 1.0	**Actel SmartDesign MSS ACE Simulation** [pdf] User Guide SmartDesign MSS ACE Simulation, SmartDesign, MSS ACE Simulation

References

- **FPGAs and PLDs \| Microchip Technology**
- **actel.com.cn**