
SmartDesign MSS

ACE Simulation



Table of Contents

Introduction	3
1 Creating the Design	5
Configuring the MSS	5
Generate the MSS - Create a Top Level SmartDesign Wrapper	7
2 Preparing the Testbench	9
Creating a Custom Testbench	9
Modifying Our BFM Script	11
Associating our Custom Testbench with our Design	12
Simulate	12
3 CAE Analog Drivers	13
Connecting Analog Ports with Verilog	13
Connecting Analog Ports with VHDL	15
A Product Support	19
Contacting the Customer Technical Support Center	19
Non-Technical Customer Service	19

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.

1 – Creating the Design

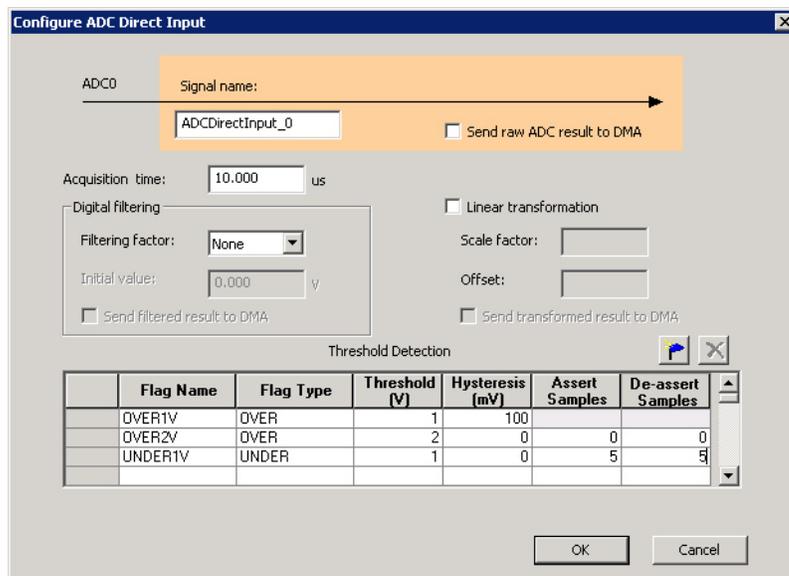
We will create a simple SmartFusion MSS and ACE configuration to demonstrate how you can simulate the ACE.

Configuring the 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 in [Figure 1-1](#) and [Figure 1-2](#).



The screenshot shows the 'Configure ADC Direct Input' dialog box. The 'Signal name' field is set to 'ADCDirectInput_0'. The 'Acquisition time' is 10.000 us. The 'Digital filtering' section shows 'Filtering factor' set to 'None' and 'Initial value' set to 0.000 v. The 'Threshold Detection' section contains a table with the following data:

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 1-1 • ADC Direct Input and Threshold Configuration

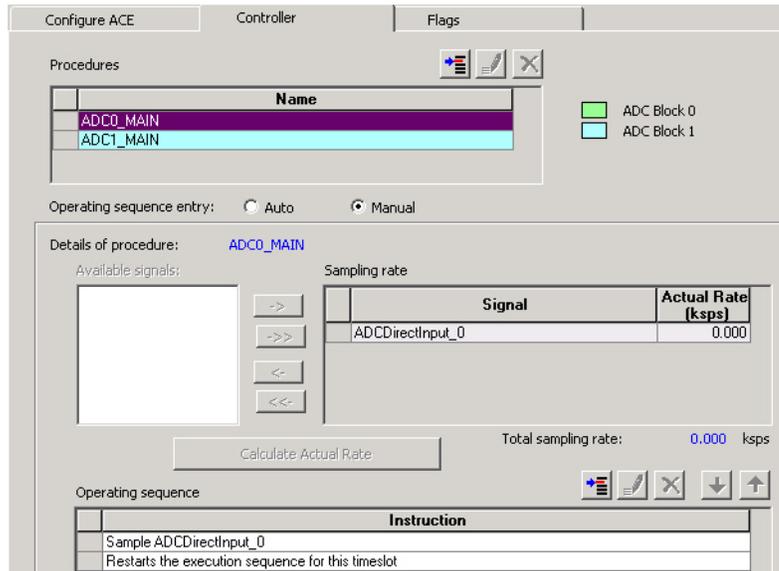


Figure 1-2 • Sampling Sequence Configuration

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 [Figure 1-3](#)).

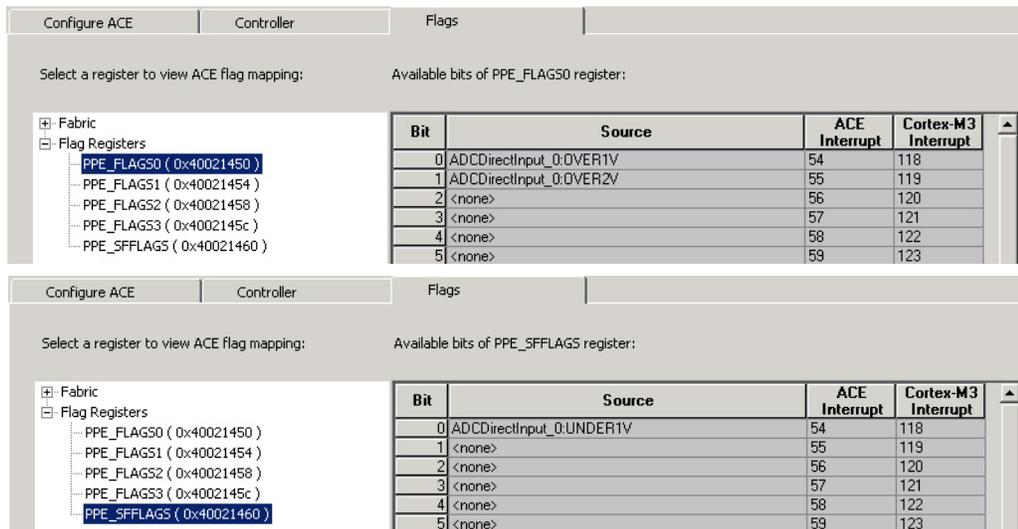


Figure 1-3 • Flag Mapping

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 Figure 1-4 after configuration:

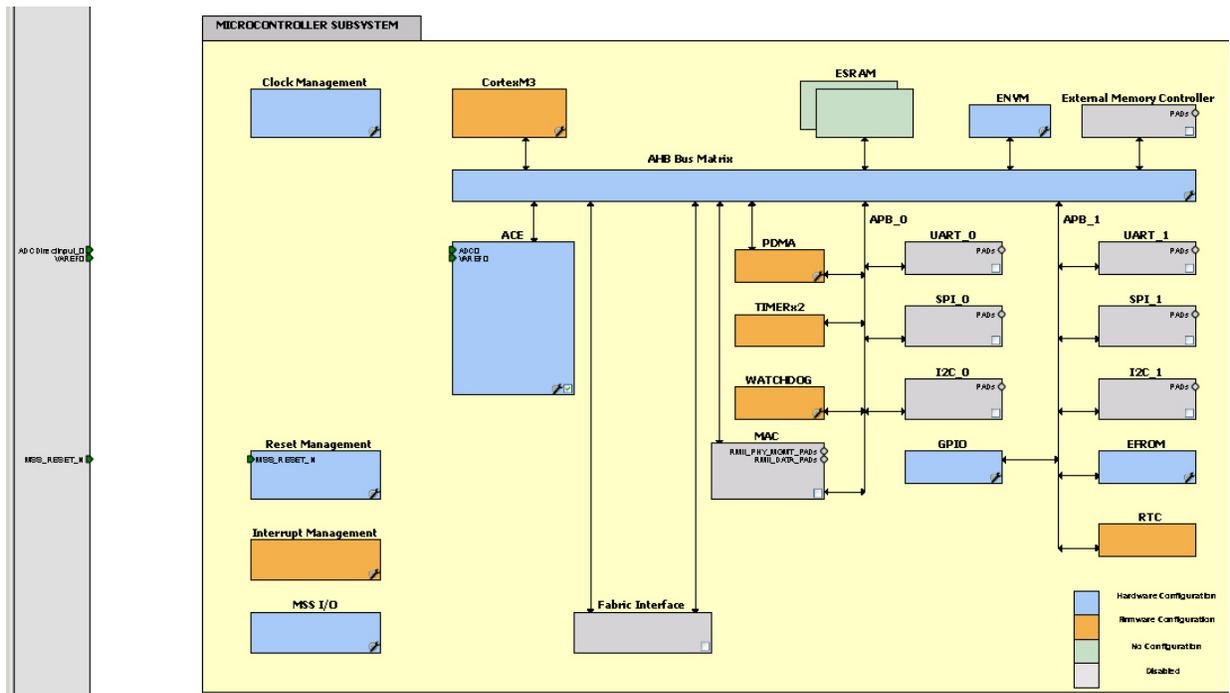


Figure 1-4 • Sample MSS Design After Configuration

Generate the MSS - Create a 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 (as shown in Figure 1-5).

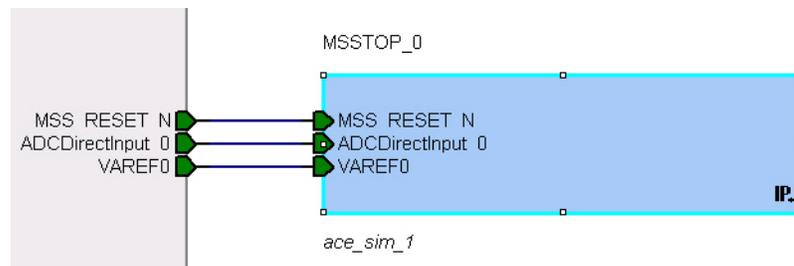


Figure 1-5 • Top-Level SmartDesign on the Canvas

2 – 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 Figure 2-1).

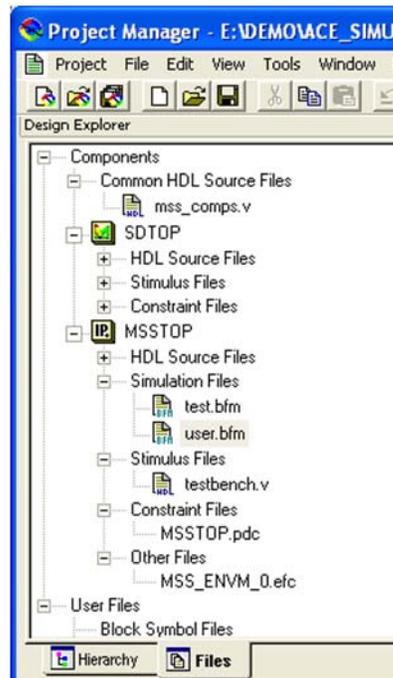


Figure 2-1 • Files Tab (File Hierarchy) in the Project Manager

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**.
Select **HDL Stimulus File**
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 code fragment in [Figure 2-2](#) 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 );
```

Figure 2-2 • Custom Testbench Code Fragment

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"](#) on page 13 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 [Actel SmartFusion Microcontroller Subsystem \(MSS\) User's Guide](#). 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 the commands shown in [Table 2-1](#).

Table 2-1 • user.bfm Script File 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** (Figure 2-3).
3. Click **OK**

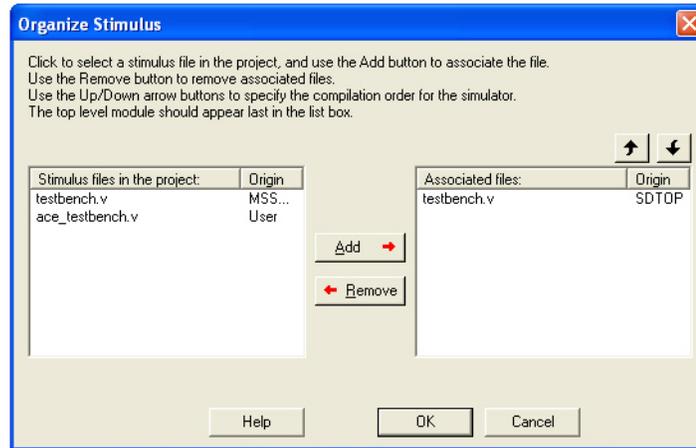


Figure 2-3 • Organize Stimulus Dialog Box

Simulate

Now we are 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.

3 – 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.

Table 3-1 shows the analog drivers that are available in Verilog.

Table 3-1 • Verilog Drivers

```
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
```

Table 3-2 demonstrates all the relevant drivers.

Table 3-2 • Driver Examples

```
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 example below illustrates the 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;           -- temperature 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 temperature 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 temperature 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_temperature_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_temperature_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;
```

A – Product Support

The Microsemi SoC Products Group backs its products with various support services including a Customer Technical Support Center and Non-Technical Customer Service. This appendix contains information about contacting the SoC Products Group and using these support services.

Contacting the Customer Technical Support Center

Microsemi staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Customer Technical Support Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

Microsemi customers can receive technical support on Microsemi SoC products by calling Technical Support Hotline anytime Monday through Friday. Customers also have the option to interactively submit and track cases online at My Cases or submit questions through email anytime during the week.

Web: www.actel.com/mycases

Phone (North America): 1.800.262.1060

Phone (International): +1 650.318.4460

Email: soc_tech@microsemi.com

ITAR Technical Support

Microsemi customers can receive ITAR technical support on Microsemi SoC products by calling ITAR Technical Support Hotline: Monday through Friday, from 9 AM to 6 PM Pacific Time. Customers also have the option to interactively submit and track cases online at My Cases or submit questions through email anytime during the week.

Web: www.actel.com/mycases

Phone (North America): 1.888.988.ITAR

Phone (International): +1 650.318.4900

Email: soc_tech_itar@microsemi.com

Non-Technical Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

Microsemi's customer service representatives are available Monday through Friday, from 8 AM to 5 PM Pacific Time, to answer non-technical questions.

Phone: +1 650.318.2470



Microsemi Corporation (NASDAQ: MSCC) offers the industry's most comprehensive portfolio of semiconductor technology. Committed to solving the most critical system challenges, Microsemi's products include high-performance, high-reliability analog and RF devices, mixed signal integrated circuits, FPGAs and customizable SoCs, and complete subsystems. Microsemi serves leading system manufacturers around the world in the defense, security, aerospace, enterprise, commercial, and industrial markets. Learn more at www.microsemi.com.

Corporate Headquarters

Microsemi Corporation
2381 Morse Avenue
Irvine, CA
92614-6233
USA
Phone 949-221-7100
Fax 949-756-0308

SoC Products Group

2061 Stierlin Court
Mountain View, CA
94043-4655
USA
Phone 650.318.4200
Fax 650.318.4600
www.actel.com

SoC Products Group (Europe)

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

SoC Products Group (Japan)

EXOS Ebisu Building 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Phone +81.03.3445.7671
Fax +81.03.3445.7668

SoC Products Group (Hong Kong)

Room 2107, China Resources Building
26 Harbour Road
Wanchai, Hong Kong
Phone +852 2185 6460
Fax +852 2185 6488