Synplify Pro ME L201609MSP1-5

Release Notes

2/2018

Microsemi
Power Matters.™

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at www.microsemi.com.

51300194-1/2.18

Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

Revision 1.0

Revision 1.0 is the first publication of this document.

Contents

1 Installation

- The Synplify Pro ME L201609MSP1-5 version can be downloaded from https://www.microsemi.com/products/fpga-soc/design-resources/design-software/synplify-pro-me#downloads
- Change the Libero SoC Tool Profile for Synthesis (Libero SoC -> Project -> Tool Profiles -> Synthesis) and point to the location of the newly-installed Synplify Pro executable.

2 Families Supported

Synplify Pro ME L201609MSP1-5 supports the following families:

- Fusion
- IGLOO
- IGLOOE
- IGLOO PLUS
- IGLOO2
- ProASIC3
- ProASIC3E
- ProASIC3L
- RTG4
- SmartFusion
- SmartFusion2
- PolarFire

3 Enhancements

The Synplify Pro ME (L2016.09MSP1-5) version has following enhancements for SmartFusion2, IGLOO2, and RTG4 families.

3.1 RTG4 - Disable safe implementation for FSMs

The following options can be used to insert safe implementation logic:

- Attribute *syn_encoding* = safe
- Attribute *syn_safe_case*

Enable the options under Implementation Options -> High Reliability -> Preserve and Decode unreachable states.

Synplify Pro L-2016.09M-SP1-5 issues the following error: "*Safe state machine option is not recommended for Microsemi RTG4 technology. To continue with safe state machine implementation, downgrade this error to warning*".

You have the option to downgrade this error to a warning message, and Synplify Pro implements the safe logic for FSMs if these options are present.

3.2 RTG4 - Write Byte-Enable Support for RAM

This feature is supported for RAMs inferred in non-low power (speed) mode.

Coding style examples for RAM Write Byte-Enable are provided below.

Example 1: RTL coding style for Single port RAM with write byte-enables

```
module ram (din, dout, addra, clk, wen1, wen2);
input [9:0] din;
input wen1;
input wen2;
input [9:0] addra;
input clk;
output reg [9:0] dout;
localparam max_depth=1024;
localparam min_width=10;
reg [9:0] taddra;
reg [min_width-1:0]  mem_ram[max_depth-1:0];
always @(posedge clk) begin
    taddra<=addra;
    if(wen1) mem_ram[taddra][4:0]<=din[4:0];
        if(wen2) mem_ram[taddra][9:5]<=din[9:5];
```

```
end
always @(posedge clk)
begin
        dout <= mem_ram[taddra];
end
endmodule
```

Resource Usage Report:

SLE 10 uses

Total Block RAMs (RAM1K18_RT): 1 of 209 (0%)

Total LUTs: 0

Example 2: RTL coding style for Two- port RAM with write byte-enable

```
module ram_wb_wen_2addr(din ,dout, addra, addrb, clk, wen);
input [17:0] din;
input [1:0] wen;
input [9:0] addra;
input [9:0] addrb;
input clk;
output reg [17:0] dout;
localparam max_depth=1024;
localparam min_width=18;
reg [9:0] taddra;
reg [9:0] taddrb;
reg [min_width-1:0]  mem_ram[max_depth-1:0];
always @(posedge clk)
begin
taddra<=addra;
taddrb<=addrb;
        if(wen[0])
                mem_ram[taddra][8:0]<=din[8:0];
        if(wen[1])
                mem_ram[taddra][17:9]<=din[17:9];
```

```
end
always @(posedge clk)
begin
        dout <= mem_ram[taddrb];
end
endmodule
```

Resource Usage Report:

SLE 10 uses

Total Block RAMs (RAM1K18_RT) : 1 of 209 (0%)

Total LUTs: 0

Example 3: VHDL RTL coding style for Two- port RAM with write byte-enable

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity test_LSRAM_1kx16 is
        port    (clk_wr         : in std_logic;
                 clk_rd         : in std_logic;
                 en1_wr                 : in std_logic;
        en2_wr          : in std_logic;
                 addr_wr                : in std_logic_vector(9 downto 0);
                 data_wr                : in std_logic_vector(15 downto 0);
                 addr_rd                : in std_logic_vector(9 downto 0);
                 data_rd                : out std_logic_vector(15 downto 0)
        );
end test_LSRAM_1kx16;
architecture behave of test_LSRAM_1kx16 is
        type mem_type is array (1023 downto 0) of std_logic_vector(15 downto 0);
        signal MEM1             : mem_type;
        signal r_addr_rd        : std_logic_vector(9 downto 0);
```

```
begin
process(clk_wr)
        begin
      if rising_edge(clk_wr) then

              if (en1_wr = '1') then
                  MEM1(CONV_INTEGER(addr_wr(9 downto 0)))(7 downto 0)<= data_wr(7
                  downto 0);
            end if;

            if (en2_wr = '1') then
                  MEM1(CONV_INTEGER(addr_wr(9 downto 0)))(15 downto 8) <= data_wr(15
                  downto 8);
            end if;
        end if;
end process;
        data_rd <= MEM1(CONV_INTEGER(r_addr_rd));
process(clk_rd)
begin
        if rising_edge(clk_rd) then
              r_addr_rd  <= addr_rd;
        end if;
end process;
end behave;
```

Resource Usage Report:

SLE 0 uses

Total Block RAMs (RAM1K18_RT) : 1 of 209 (0%)

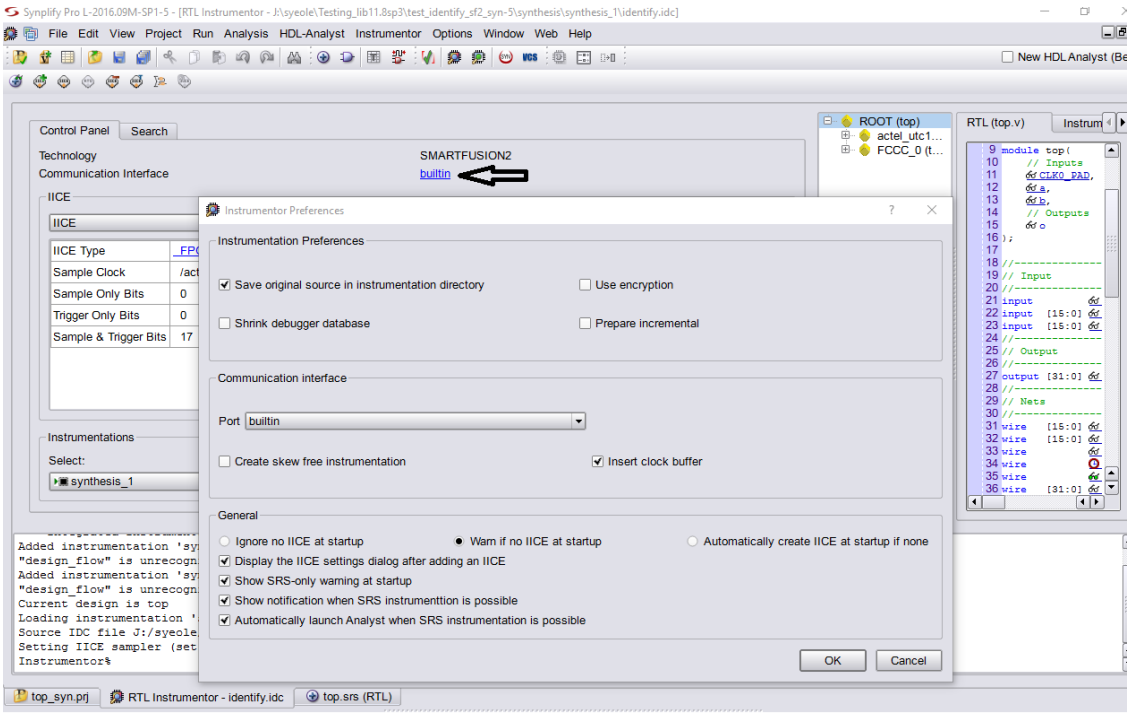
Total LUTs: 0

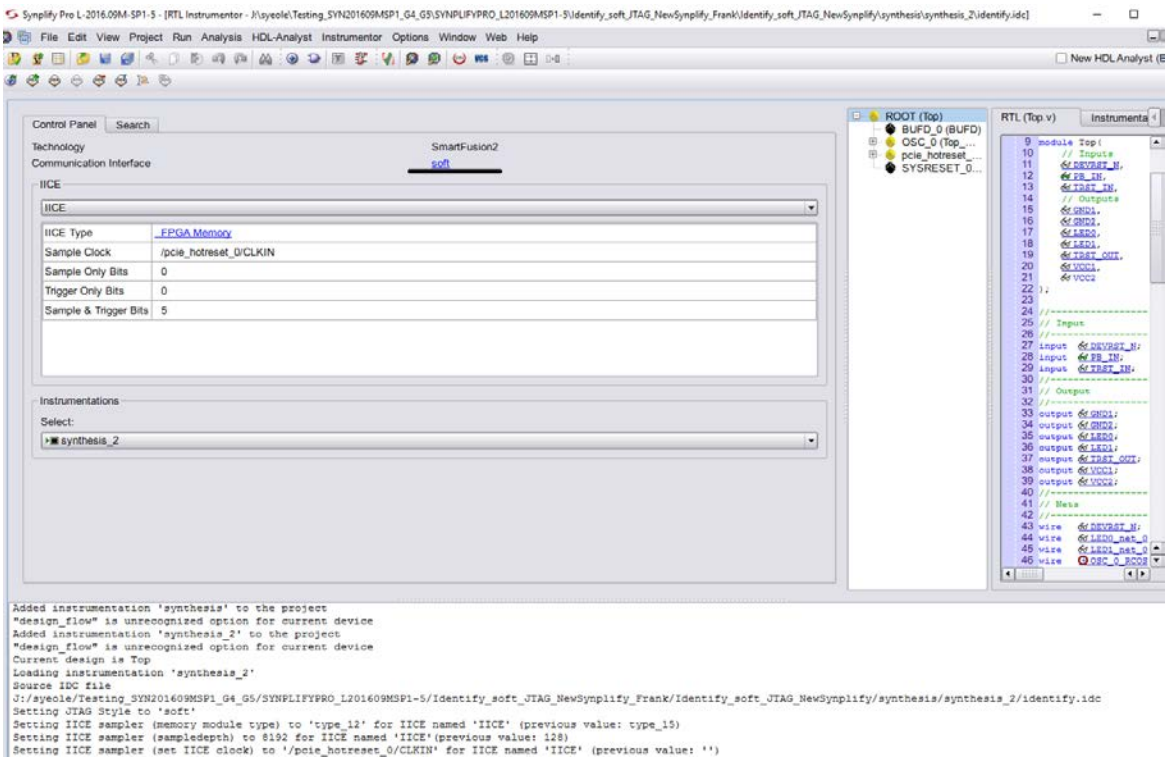
3.3 RTG4 - Updated Timing Models

Synplify Pro L-2016.09M-SP1-5 updates the timing models for RTG4 – cell delay, net delay models and carry-chain paths.

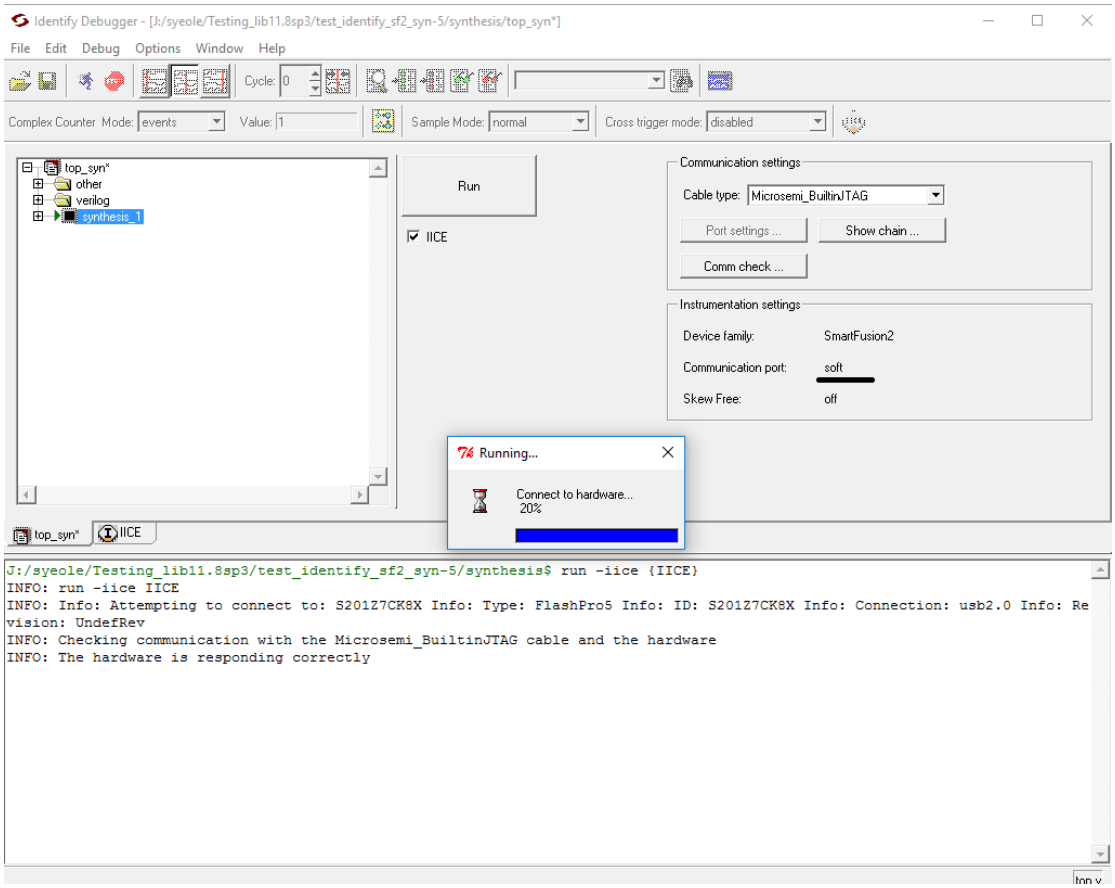
3.4 SmartFusion2, IGLOO2, RTG4 - Soft JTAG Controller feature in Identify Instrumentor

- The “soft” communication interface feature is fixed.
- Users can select through the Identify Instrumentor integrated within Synplify Pro as shown in the following screenshots:





- In Identify Debugger, the communication port selected in Instrumenter shows as “soft”



For details about this feature, refer to the Synplify Pro and Identify User Guides:

Identify -> doc -> identify_instrumentor_user_guide.pdf

Identify -> doc -> identify_debugger_user_guide.pdf

For using Identify Instrumentor and Identify Debugger with Libero SoC, refer to the tutorial at the following link:

https://www.microsemi.com/products/fpga-soc/design-resources/design-software/libero-soc#documents

4 Resolved Issues

4.1 SmartFusion2, IGLOO2, and RTG4 - Logical Bug with inference of pipelined wide multipliers into MATH blocks

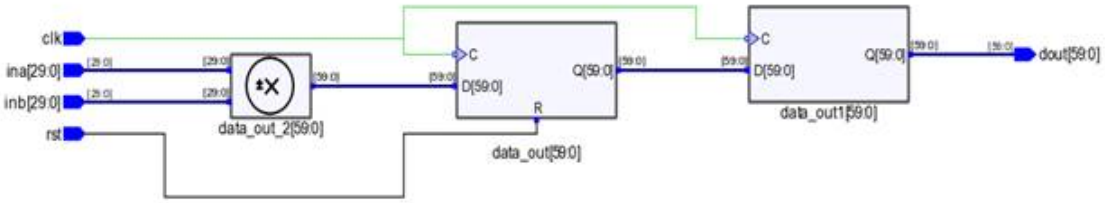
Testcase Scenario:

- Two stages of registers.
- The first stage of registers has an async or sync reset, but the second stage does not.
- Width of ina > 18 or Width of inb > 18.

In the above test scenario, Synplify Pro infers MACC block by packing registers.

Issue:

- Reset signal was left dangling and was not connected to the MACC block.



4.2 Other resolved issues

CASE	Description
493642-2121758499	Synplify does not issue warning
493642-2275822814	Synplify Pro ME L2016.09M-2 crash
493642-2329166944	Synplify Pro L-2016.09M-2 prompts warning Pure - Impure Function but J-2015.03M SP1-2 does not give warning
493642-2352648113	Internal Error in m_proasic.exe
-	For the RTG4 family, the async_globalthreshold should be set 12 by default
-	Synplify creates RGRESETs when FSM compiler option is checked for RTG4

-	Synplify Pro infers RAM1K18_RT with unsupported WMODE =01 with ECC=1
-	Synplify Pro crashes with Compiler Error for vhdl design
-	The total global count is "8" for all dies, which is incorrect.
-	Synplify Pro (embedded with Libero SoC v11.7) displays the unsupported Operating Condition for SmartFusion2/ IGLOO2/RTG4 devices
-	Need explanation for SRST_N absorption across hierarchy
-	Synplify Pro Online Help claims register initial value support
-	Synplify suboptimal 32x32 Mult
-	Forward-annotated SDF is not supported for ProASIC3 and other families and needs to be removed from doc.

5 Known Issues

5.1 Synplify Pro error: library Fusion not found

Issue: There are missing lines related to the Fusion library from the location.map from Synplify Pro installation folder.

Workaround: Updated files can be patched locally upon request.

5.2 High reliability option in Synplify Pro is not inferring the state machine correctly

Issue: With High reliability option on, Synplify Pro does not infer state machines from the state machine coding in the correct way.

Workaround: Use the attribute syn_state_machine in the case statement code as below:
attribute syn_state_machine : boolean;
attribute syn_state_machine of state : signal is true;

5.3 Warning message: "@W: CL269 : State error detection not built"

Reason: *Refer to Synplify Pro Help for CL269 warning message.*

In safe mode, the compiler generates a state error detection component for all case statements used to synthesize the state machine logic. If the component is removed, this warning is generated for you to confirm the following:

1. A state machine was not inferred for a particular case statement.
2. A state machine was inferred from a case statement, but the case statement is missing an others clause (VHDL) or default clause (Verilog).

Action: Check for the correct intended behavior. In the first condition above, verify whether the logic should not be a state machine or if the compiler was unable to extract it. For the second scenario, you may need to add an others or default clause to the source code so the compiler knows how to handle a bad state.

Workaround: If you believe a state machine was not inferred for a particular case statement, analyze the RTL to understand why the logic should not be a state machine. If the intended behavior is correct, ignore the warning message. Otherwise, modify the RTL so that a state machine is inferred.