Creating Multiple Instances of an SOPC System in a Quartus II Project

(Workaround for JTAG Node ID conflicts)

Author: Jakob Jones

Table of Contents

Table of Figures

Abstract

This paper presents a method for working around the problem of conflicting JTAG node IDs when creating multiple instances of an Altera SOPC system within a Quartus II project. The paper describes how the project can be successfully compiled in Quartus II as well as how debugging can be achieved on a unique NIOS II processor instance via the NIOS II IDE. This information is current as of Quartus II version 6.0. Design examples are presented.

Introduction

The Altera SOPC Builder can be a powerful tool in Altera FPGA firmware designs. SOPC Builder allows easy creation of systems based on the Altera NIOS II processor along with any other entity which conforms to the Avalon bus standard. Currently, SOPC Builder has some limitations with regards to the JTAG scan chain.

Certain design entities within the SOPC system make use of the JTAG chain. Examples of this are the JTAG debug module connected to the NIOS II processor and the JTAG UART core. Every JTAG device (hardware or firmware) connected to the scan chain requires a unique JTAG node ID. When the SOPC system is generated, Verilog HDL code is generated for the individual components within the SOPC system. The JTAG node ID assignments for each entity attached to the JTAG chain are made within the generated Verilog code.

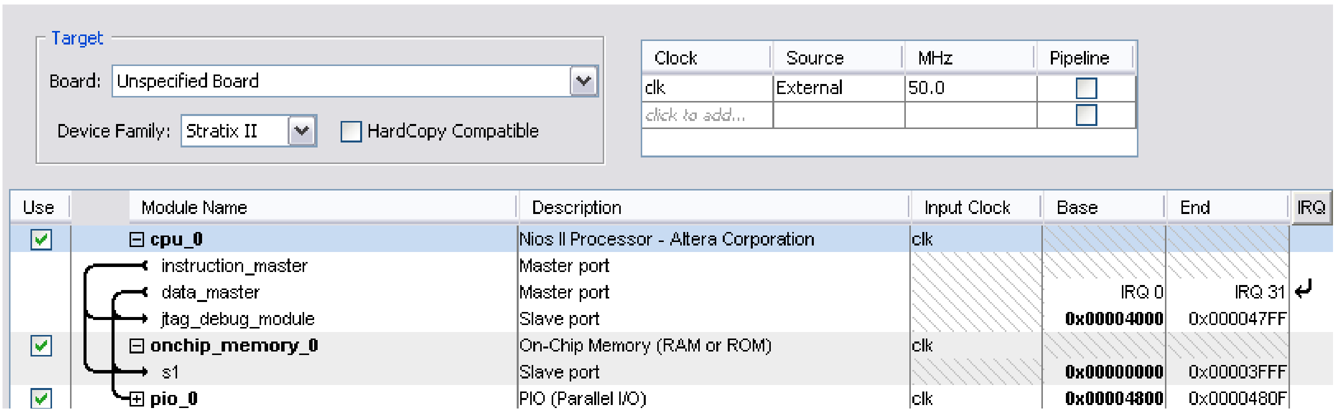
When the developer attempts to create multiple instances of the SOPC system in his/her firmware, the project fails to pass the “Analysis/Synthesis” step due to the conflicting node ID assignments between instances of the SOPC system.

This paper will explain how the JTAG node ID values are assigned, how to resolve the conflicts such that the project can be successfully compiled in Quartus II, and how to debug a unique instance of a NIOS II processor from the NIOS II IDE.

Resolving JTAG Node ID Conflicts

Design Example 1

Consider the example SOPC system found in Figure 1. The systems consists of a NIOS II processor with a JTAG debug module, some onchip memory (made of M4K blocks), and a single bit, output only, PIO. The PIO output will be connected to an LED in the top level Quartus project.



Target

Board: Unspecified Board

Device Family: Stratix II ☐ HardCopy Compatible

Clock	Source	MHz	Pipeline
clk	External	50.0	☐
click to add...			☐

Use	Module Name	Description	Input Clock	Base	End	IRQ
✓	⊟ cpu_0	Nios II Processor - Altera Corporation	clk			
	instruction_master	Master port				
	data_master	Master port		IRQ 0	IRQ 31	↵
	jtag_debug_module	Slave port		0x00004000	0x000047FF	
✓	⊟ onchip_memory_0	On-Chip Memory (RAM or ROM)	clk			
	s1	Slave port		0x00000000	0x00003FFF	
✓	⊞ pio_0	PIO (Parallel I/O)	clk	0x00004800	0x0000480F	

Figure 1: Example SOPC System.

When this SOPC system is generated, a file with the extension of .ptf will be created which describes the system. In addition, Verilog HDL code will be generated to describe the RTL of the system. The .ptf file will continue to be used by SOPC builder and is also used by the NIOS II IDE. The generated Verilog code is used by Quartus II to synthesize the logic for the system.

JTAG Node ID Assignment in <cpu_name>_jtag_debug_module.v file

When the SOPC system is generated, a Verilog file named <cpu_name>_jtag_debug_module.v is created (<cpu_name> refers to the name of the processor in the system. In this example <cpu_name> is "cpu_0"). This file contains a parameter statement defining a parameter `SLD_NODE_INFO` as shown below.

```
parameter SLD_NODE_INFO = 286279168;
```

This parameter statement defines the JTAG node ID for the JTAG debug module. But where did this ID number come from?

JTAG Node ID Assignment in .ptf File

Listing 1 on page 5 shows a snippet of the .ptf file that was generated by the SOPC Builder. The snippet shows two important assignments (highlighted in the listing). These assignments are `JTAG_Hub_Base_Id` and `JTAG_Hub_Instance_Id`. In our example, these are assigned the values of 1118278 and 0 respectively. The value of the `SLD_NODE_INFO` parameter in the Verilog HDL file is derived from these two assignments. Specifically SLD_NODE_INFO is assigned the value of JTAG_Hub_Instance_Id added to the value of JTAG_Hub_Instance_Id shifted to the left 8 bits.

```
SLD_NODE_INFO = (JTAG_Hub_Base_Id << 8) + JTAG_Hub_Instance_Id
```

```
SYSTEM_BUILDER_INFO
         {
            Read_Wait_States = "1";
            Write_Wait_States = "1";
            Register_Incoming_Signals = "1";
            Bus_Type = "avalon";
            Data_Width = "32";
            Address_Width = "9";
            Accepts_Internal_Connections = "1";
            Requires_Internal_Connections = "instruction_master,data_master";
            Accepts_External_Connections = "0";
            Is_Enabled = "1";
            Address_Alignment = "dynamic";
            Base_Address = "0x00004000";
            Is_Memory_Device = "1";
            Is_Readable = "1";
            Is_Writeable = "1";
            Is_Printable_Device = "0";
            Is_Big_Endian = "0";
            Uses_Tri_State_Data_Bus = "0";
            Has_IRQ = "0";
            JTAG_Hub_Base_Id = "1118278";
            JTAG_Hub_Instance_Id = "0";
            MASTERED_BY cpu_0/instruction_master
            {
```

Listing 1: JTAG Node ID Assignment in Generated .ptf File

Top Level Quartus II Project (Creating Multiple SOPC Instances).

Now that we have created our SOPC system, we can create multiple instances of it in our top level Quartus project. Listing 2 shows the top level Verilog code for our project. The top level project as one clock input (clk), one reset input (reset_n), and four led outputs (led_out). In this design example, the SOPC system in Figure 1 was given the name of sopc_core. Here we created 3 instances of our SOPC system and connected each of the PIO outputs to a led output pin. We also created a PLL to convert the 100MHz input clock to a 50MHz clock for the SOPC systems.

```
module dmate2_multi_core_top(
        clk,                          //input clock 100MHz
        reset_n,                      //input reset button
        led_out                       //output to leds (active low)
        );

input                 clk;
input                 reset_n;
output  [3:0]   led_out;

wire sopc_clk;                //50MHz clock to SOPC systems
//Instantiate PLL to convert 100MHz input clock to 50MHz
//SOPC clock.
sopc_pll sopc_pll_inst(
        .inclk0(clk),
        .c0(sopc_clk)
        );

//We now create 3 instances of one of the SOPC systems
sopc_core sopc_core_0(
        // 1) global signals:
    .clk(sopc_clk),
    .reset_n(reset_n),
    // the_pio_0
    .out_port_from_the_pio_0(led_out[0])
    );

sopc_core sopc_core_1(
        // 1) global signals:
    .clk(sopc_clk),
    .reset_n(reset_n),
    // the_pio_0
    .out_port_from_the_pio_0(led_out[1])
    );

sopc_core sopc_core_2(
        // 1) global signals:
    .clk(sopc_clk),
    .reset_n(reset_n),
    // the_pio_0
    .out_port_from_the_pio_0(led_out[2])
    );

endmodule
```

Listing 2: Top level Verilog module.

JTAG Node ID Conflicts

We can now attempt to compile our top level project in Quartus. We click the compile button and about 10 seconds into the Analysis & Synthesis step the compilation fails with the errors shown in Listing 3. The errors are straightforward. Because all of the instances of our SOPC system are using the same HDL code which contains the JTAG Node ID assignment, they are all getting the same Node ID which isn't acceptable to Quartus II.

```
Error: Debug node with entity name "cpu_0_jtag_debug_module" and instance name
"sopc_core:sopc_core_0|cpu_0:the_cpu_0|cpu_0_nios2_oci:the_cpu_0_nios2_oci|
cpu_0_jtag_debug_module_wrapper:the_cpu_0_jtag_debug_module_wrapper|
cpu_0_jtag_debug_module:the_cpu_0_jtag_debug_module1" has duplicate ID 286279168 -- node ID is
used by another node

Error: Debug node with entity name "cpu_0_jtag_debug_module" and instance name
"sopc_core:sopc_core_0|cpu_0:the_cpu_0|cpu_0_nios2_oci:the_cpu_0_nios2_oci|
cpu_0_jtag_debug_module_wrapper:the_cpu_0_jtag_debug_module_wrapper|
cpu_0_jtag_debug_module:the_cpu_0_jtag_debug_module1" has duplicate ID 286279168 -- node ID is
used by another node

Error: Debug node with entity name "cpu_0_jtag_debug_module" and instance name
"sopc_core:sopc_core_1|cpu_0:the_cpu_0|cpu_0_nios2_oci:the_cpu_0_nios2_oci|
cpu_0_jtag_debug_module_wrapper:the_cpu_0_jtag_debug_module_wrapper|
cpu_0_jtag_debug_module:the_cpu_0_jtag_debug_module1" has duplicate ID 286279168 -- node ID is
used by another node

Error: Two instances "sopc_core:sopc_core_0|cpu_0:the_cpu_0|cpu_0_nios2_oci:the_cpu_0_nios2_oci|
cpu_0_jtag_debug_module_wrapper:the_cpu_0_jtag_debug_module_wrapper|
cpu_0_jtag_debug_module:the_cpu_0_jtag_debug_module1" and "sopc_core:sopc_core_1|cpu_0:the_cpu_0|
cpu_0_nios2_oci:the_cpu_0_nios2_oci|
cpu_0_jtag_debug_module_wrapper:the_cpu_0_jtag_debug_module_wrapper|
cpu_0_jtag_debug_module:the_cpu_0_jtag_debug_module1" of the same debug node with entity name
"cpu_0_jtag_debug_module" have the same ID 286279168.

Error: Two instances "sopc_core:sopc_core_0|cpu_0:the_cpu_0|cpu_0_nios2_oci:the_cpu_0_nios2_oci|
cpu_0_jtag_debug_module_wrapper:the_cpu_0_jtag_debug_module_wrapper|
cpu_0_jtag_debug_module:the_cpu_0_jtag_debug_module1" and "sopc_core:sopc_core_2|cpu_0:the_cpu_0|
cpu_0_nios2_oci:the_cpu_0_nios2_oci|
cpu_0_jtag_debug_module_wrapper:the_cpu_0_jtag_debug_module_wrapper|
cpu_0_jtag_debug_module:the_cpu_0_jtag_debug_module1" of the same debug node with entity name
"cpu_0_jtag_debug_module" have the same ID 286279168.

Error: Two instances "sopc_core:sopc_core_1|cpu_0:the_cpu_0|cpu_0_nios2_oci:the_cpu_0_nios2_oci|
cpu_0_jtag_debug_module_wrapper:the_cpu_0_jtag_debug_module_wrapper|
cpu_0_jtag_debug_module:the_cpu_0_jtag_debug_module1" and "sopc_core:sopc_core_2|cpu_0:the_cpu_0|
cpu_0_nios2_oci:the_cpu_0_nios2_oci|
cpu_0_jtag_debug_module_wrapper:the_cpu_0_jtag_debug_module_wrapper|
cpu_0_jtag_debug_module:the_cpu_0_jtag_debug_module1" of the same debug node with entity name
"cpu_0_jtag_debug_module" have the same ID 286279168.
```

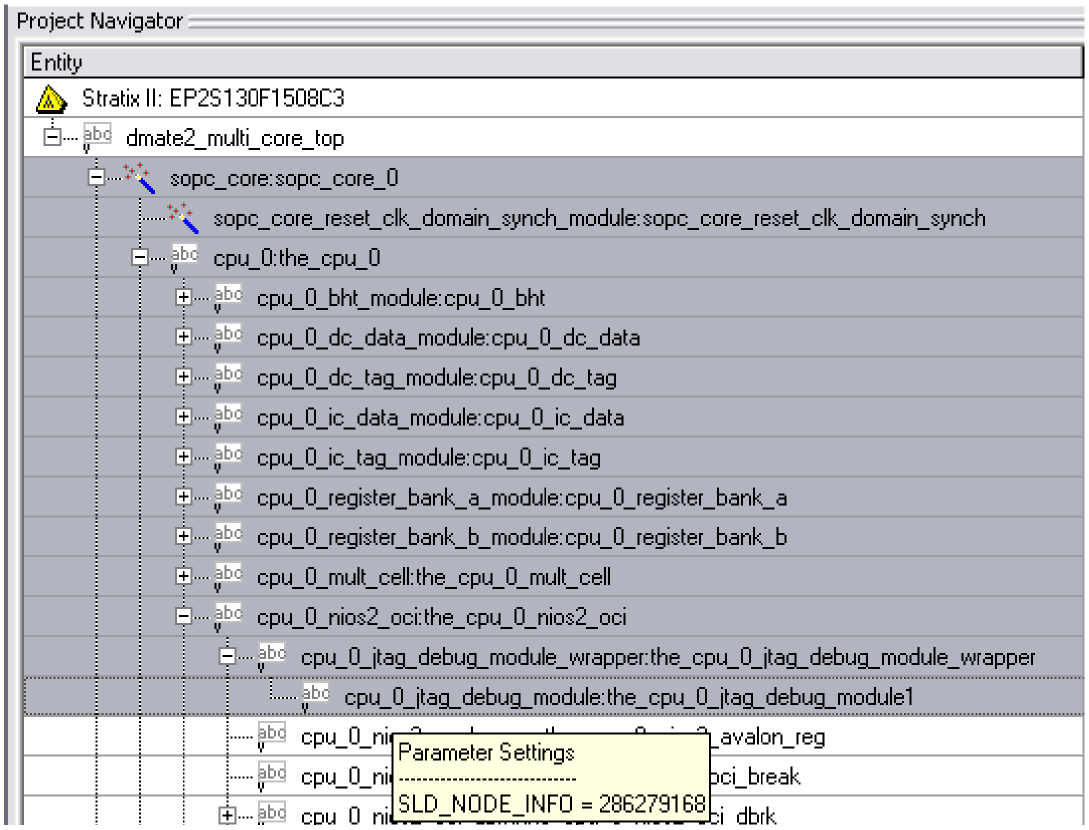
Listing 3: Generated compilation errors due to JTAG Node ID conflicts.

Resolving JTAG Node ID Conflicts Using The Assignment Editor

Because the JTAG node ID assignments are simply parameter statements in HDL, we can use the Assignment Editor in Quartus II to change them for each design entity. To do this, we can use the Quartus II Project Navigator to browse to the design entity where the JTAG debug module for the particular instance of our NIOS II processor is located.

The location of the JTAG debug module in the design hierarchy for the first instance of our SOPC system is shown in Figure 2. The names of these entities will obviously change based on the name of the SOPC system and the name of the processor in the system. When you have correctly located the JTAG debug module in the design hierarchy you should get the yellow hint box showing the value of the SLD_NODE_INFO parameter. Once you have found the entity, perform the following:

Right-click->Locate->Locate In Assignment Editor.

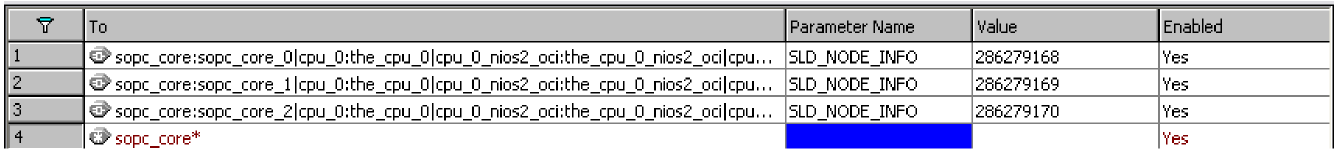


```
Project Navigator
Entity
Stratix II: EP2S130F1508C3
  dmate2_multi_core_top
    sopc_core:sopc_core_0
      sopc_core_reset_clk_domain_synch_module:sopc_core_reset_clk_domain_synch
      cpu_0:the_cpu_0
        cpu_0_bht_module:cpu_0_bht
        cpu_0_dc_data_module:cpu_0_dc_data
        cpu_0_dc_tag_module:cpu_0_dc_tag
        cpu_0_ic_data_module:cpu_0_ic_data
        cpu_0_ic_tag_module:cpu_0_ic_tag
        cpu_0_register_bank_a_module:cpu_0_register_bank_a
        cpu_0_register_bank_b_module:cpu_0_register_bank_b
        cpu_0_mult_cell:the_cpu_0_mult_cell
        cpu_0_nios2_oci:the_cpu_0_nios2_oci
          cpu_0_jtag_debug_module_wrapper:the_cpu_0_jtag_debug_module_wrapper
            cpu_0_jtag_debug_module:the_cpu_0_jtag_debug_module1
          cpu_0_ni                              avalon_reg
          cpu_0_ni                              oci_break
          cpu_0_ni                              ci_dbrk

Parameter Settings
----------------------------
SLD_NODE_INFO = 286279168
```

Figure 2: Location of JTAG Debug Module in Design Hierarchy.

In the assignment editor we can now change the parameter assignment for that instance's parameter as shown in Figure 3. Notice that in the figure shown we have made assignments for all three instances of the JTAG debug module in our project. We have left one of the assignments the same and changed the other two. These assignments could also easily be made from a TCL script.



	To	Parameter Name	Value	Enabled
1	sopc_core:sopc_core_0|cpu_0:the_cpu_0|cpu_0_nios2_oci:the_cpu_0_nios2_oci|cpu...	SLD_NODE_INFO	286279168	Yes
2	sopc_core:sopc_core_1|cpu_0:the_cpu_0|cpu_0_nios2_oci:the_cpu_0_nios2_oci|cpu...	SLD_NODE_INFO	286279169	Yes
3	sopc_core:sopc_core_2|cpu_0:the_cpu_0|cpu_0_nios2_oci:the_cpu_0_nios2_oci|cpu...	SLD_NODE_INFO	286279170	Yes
4	sopc_core*			Yes

Figure 3: Modified JTAG Node ID Assignments in Assignment Editor.

In this example we have simply incremented to get the other two node IDs. This can get you into trouble in some cases. For example, if I generate another SOPC system to be used in this project it will likely take the JTAG node ID of 286279169 which I have designated for another use. It is recommended that you instead assign a random value to the other node IDs. However, you SHOULD NOT CHANGE THE BASE ID VALUE. This is the JTAG_Hub_Base_Id value from the .ptf file. So the valid numbers that we can choose from in our example are between 286279168 and 286279423.

Now with new parameter assignments made, the project compiles successfully and runs on the target hardware.

Debugging Multiple Instances of a NIOS II Processor Using the NIOS II IDE

In the previous section we used a design example to illustrate how we could resolve JTAG Node ID conflicts when creating multiple instances of an SOPC system in a Quartus II project. In this section we will look at how to debug multiple instances of a NIOS II processor using the NIOS II IDE. The examples we use here are for the example design project in the previous section. These examples assume that the reader is already familiar with how to debug multiple projects simultaneously from within the NIOS II IDE. If this is not the case, please see the documentation at Altera's website.

In general, while debugging software, there is really no need to debug more than one instance of a NIOS II processor. However, suppose the multiple instantiations of your SOPC system pass data back and forth to each other through some type of mailbox and you wish to observe this. Or suppose each instance of your SOPC system is connected to some piece of hardware on your board and one particular instance isn't working properly. These are some examples of when you may wish to target a specific instance or multiple instances of an SOPC system for debugging.

How NIOS II IDE Selects a Processor in the System for Debugging

Currently the NIOS II IDE does not provide any built-in method for targeting a specific instance of your SOPC system for debugging. Hopefully this will change in the future. For now, there is only one simple rule to remember with regards to how the IDE selects a processor for debugging.

The target processor that NIOS II IDE targets for debugging is the one who's JTAG debug module has a node ID identical to that described in the .ptf file generated by SOPC Builder.

That's it. So with that in mind, there are several ways that you could target a specific processor:

1) Modify the .ptf file each time you wish to target a different processor.
2) Create a copy of the .ptf file with unique JTAG Node ID information for each processor you wish to target.

Here are some notes on these options. In option 1, if you modify the generated .ptf file, that change **will not** be overwritten if the SOPC system is modified and regenerated. That's kind of nice to know. In fact, it will retain your change and the newly generated HDL code will also reflect your changes. Using this method you will only be able to debug one processor at a time.

In option 2, obviously the copies you made of the .ptf file will be out of date if the original SOPC system is regenerated. You should probably create a script or batch file to copy the new .ptf file and make the changes for you. Using this method you can simultaneously debug as many instances of your SOPC system as you wish.

Example Debug Configuration of Multiple SOPC System Instances

For this example, we will use option 2 listed above. We will create a copy of the generated .ptf file for each processor we wish to debug. Then we will set up the NIOS IDE to debug all three instances of our SOPC system simultaneously.

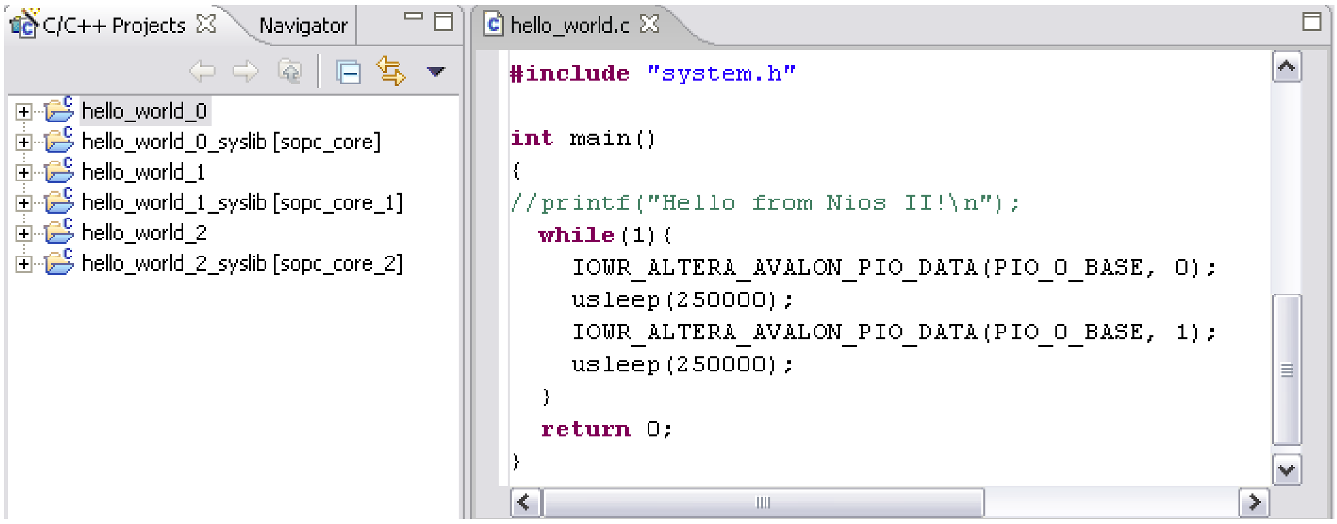
The original .ptf file that SOPC Builder generated for us was named sopc_core.ptf. We will create two copies of this file (sopc_core_1.ptf and sopc_core_2.ptf) and modify the JTAG node IDs found in those files. Listing 4 shows the newly assigned values in each .ptf file. Note that these values were selected to match the SLD_NODE_INFO assignments made for each SOPC instance in the Assignment Editor in the previous section. These values were 286279168, 286279169, and 286279170 respectively.

```
JTAG_Hub_Base_Id ="1118278";   JTAG_Hub_Base_Id ="1118278";   JTAG_Hub_Base_Id ="1118278";
JTAG_Hub_Instance_Id = "0";    JTAG_Hub_Instance_Id = "1";    JTAG_Hub_Instance_Id = "2";
```

Listing 4: Modified JTAG Node ID Values in .ptf Files for Debugging. Values for sopc_core.ptf, sopc_core_1.ptf, and sopc_core_2.ptf respectively.

Now that we have created our unique .ptf files, we can start the NIOS II IDE. It is important to note that the NIOS II IDE only reads these node ID values when it is first started. **Changes made to the node ID values within the .ptf files while the IDE is open will not be recognized by the IDE.**

In the NIOS II IDE, we now create a system library project for each instance of the processor that we wish to debug. When creating the system library, we point to the .ptf file for the SOPC instance we wish to target. I then create a simple "Hello World" project for each instance. Figure 4 shows a snapshot of the NIOS II IDE with my three projects and system libraries as well as the main function routine that will be executed on each processor. It is a simple infinite loop that blinks the LED attached to the PIO.



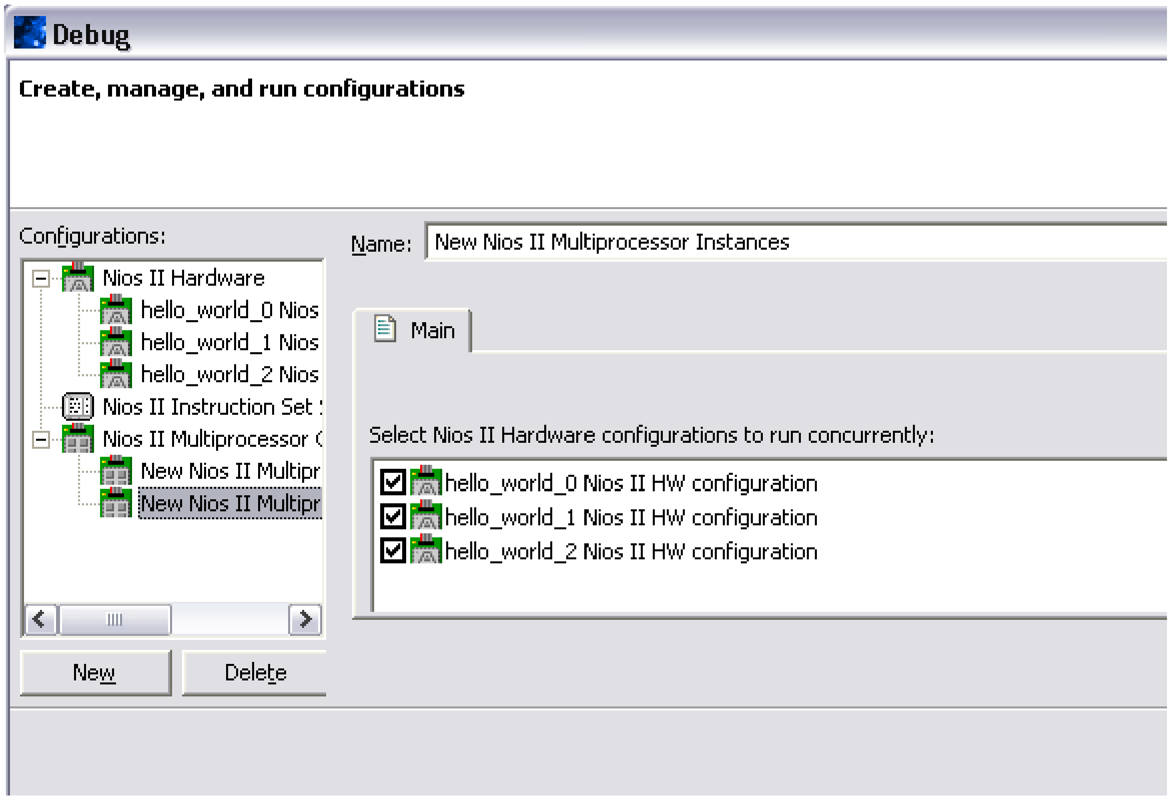
hello_world_0
hello_world_0_syslib [sopc_core]
hello_world_1
hello_world_1_syslib [sopc_core_1]
hello_world_2
hello_world_2_syslib [sopc_core_2]

```
#include "system.h"

int main()
{
//printf("Hello from Nios II!\n");
  while(1){
    IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE, 0);
    usleep(250000);
    IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE, 1);
    usleep(250000);
  }
  return 0;
}
```

Figure 4: Screenshot of NIOS II IDE with projects for each SOPC instance to be debugged.

Once we have our three projects created, we create debug configurations for each one. Then we create a Multiprocessor debug configuration to debug all three at the same time as shown in Figure 5.


Debug

Create, manage, and run configurations

Configurations:

- Nios II Hardware
  - hello_world_0 Nios
  - hello_world_1 Nios
  - hello_world_2 Nios
- Nios II Instruction Set :
- Nios II Multiprocessor (
  - New Nios II Multipr
  - New Nios II Multipr

Name: New Nios II Multiprocessor Instances

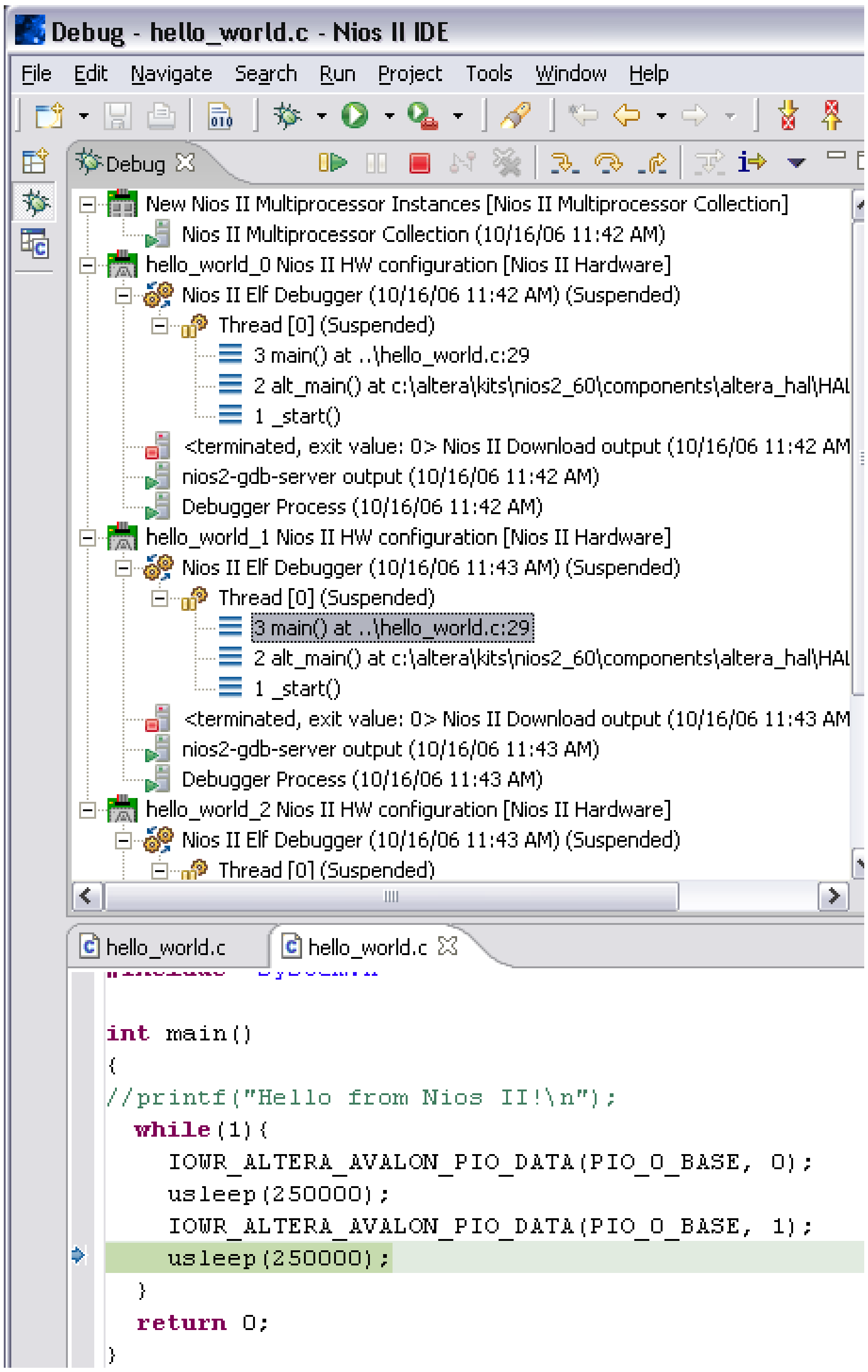
Main

Select Nios II Hardware configurations to run concurrently:

- ✓ hello_world_0 Nios II HW configuration
- ✓ hello_world_1 Nios II HW configuration
- ✓ hello_world_2 Nios II HW configuration

Figure 5: Creating a multiprocessor debug configuration.

Now we simply debug our multiprocessor configuration. The IDE will launch the debugger, download the code to each instance of the NIOS II processor, and break at the entry to the main function. Figure 6 shows the IDE in debug perspective debugging all three instances of our SOPC system simultaneously. We can individually step through the code on each processor separately.


Debug - hello_world.c - Nios II IDE
File Edit Navigate Search Run Project Tools Window Help
Debug
hello_world.c
hello_world.c

```
New Nios II Multiprocessor Instances [Nios II Multiprocessor Collection]
    Nios II Multiprocessor Collection (10/16/06 11:42 AM)
hello_world_0 Nios II HW configuration [Nios II Hardware]
    Nios II Elf Debugger (10/16/06 11:42 AM) (Suspended)
        Thread [0] (Suspended)
            3 main() at ..\hello_world.c:29
            2 alt_main() at c:\altera\kits\nios2_60\components\altera_hal\HAL
            1 _start()
    <terminated, exit value: 0> Nios II Download output (10/16/06 11:42 AM
    nios2-gdb-server output (10/16/06 11:42 AM)
    Debugger Process (10/16/06 11:42 AM)
hello_world_1 Nios II HW configuration [Nios II Hardware]
    Nios II Elf Debugger (10/16/06 11:43 AM) (Suspended)
        Thread [0] (Suspended)
            3 main() at ..\hello_world.c:29
            2 alt_main() at c:\altera\kits\nios2_60\components\altera_hal\HAL
            1 _start()
    <terminated, exit value: 0> Nios II Download output (10/16/06 11:43 AM
    nios2-gdb-server output (10/16/06 11:43 AM)
    Debugger Process (10/16/06 11:43 AM)
hello_world_2 Nios II HW configuration [Nios II Hardware]
    Nios II Elf Debugger (10/16/06 11:43 AM) (Suspended)
        Thread [0] (Suspended)
```

```
int main()
{
//printf("Hello from Nios II!\n");
  while(1){
    IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE, 0);
    usleep(250000);
    IOWR_ALTERA_AVALON_PIO_DATA(PIO_0_BASE, 1);
    usleep(250000);
  }
  return 0;
}
```

Figure 6: Debugging Multiple SOPC Instances Simultaneously.

Conclusion

This paper has presented a workaround for the JTAG node ID conflict that occurs when attempting to create multiple instances of an SOPC system in a Quartus II project. We have also shown that it is possible to debug any one or all of the instances through the NIOS II IDE. This workaround has limitations. It is hoped that future versions of SOPC Builder, and Quartus II will have native support for managing multiple instances of an SOPC system.