

NIOS II Custom Instruction for MAX 10 FPGA Evaluation Kit

©2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

This document shows the Custom instruction for NIOS II design example for the MAX 10 FPGA Evaluation Kit.

About Custom Instructions in NIOS II:

When a design includes an Altera Nios II embedded processor, the design can accelerate time-critical software algorithms by adding custom instructions to the Nios II processor instruction set. Custom instructions allow the user to reduce a complex sequence of standard instructions to a single instruction implemented in hardware. This feature can be used for a variety of applications, for example, to optimize Software inner loops for digital signal processing (DSP), packet header processing, and computation-intensive applications. In Qsys, each custom instruction is a separate component in the Qsys system. The user can add as many as 256 custom instructions to your system.

Refer to <u>Nios II Custom Instruction User Guide</u> for further information.

About this Design Example

This design example shows how to implement the cyclic redundancy check (CRC) algorithm as a Nios II custom instruction. The CRC algorithm detects the corruption of data during transmission. The CRC calculation consists of an iterative algorithm involving XOR and shift operations. These operations are carried out concurrently in hardware and iteratively in software. Since the operations are carried out concurrently, the execution is much faster in hardware. This example demonstrates the way to implement an extended multi-cycle Nios II custom instruction.

This design Example performs the following:

- 1) Computation of CRC using the custom instruction
- 2) Computation of CRC using a software C code.
- 3) Computation of CRC using a optimized software C code.

Steps to Run the Program

The steps to run the Custom Instruction for the MAX 10 Eval Kit are:

- 1. Extract all the files from the custom_inst.par by following the instructions on the design store. The extracted files have the following the file structure:
 - i) Crc_hw folder consists of the 2 Verilog files CRC_Component.v and CRC_Custom_Instruction.v for the custom instruction.
 - ii) The master_image folder consists of the custom_inst.sof and custom_inst.pof file which can be directly used by the user to program the board. It also contains he .hex file which can be used to directly run the program from the nios terminal.
 - iii) The software/src folder contains all the .c files
 crc_main.c : is the main file which calls all the other files.
 crc.c: contains the software implementation and the optimized software implementation of
 CRC function.
 ci_crc.c: is the .c file for executing the built in function for the custom instruction.

ci_crc.h: hex file related to custom instruction

crc.h: hex file related to the software implementation.

- iv) It also contains platform/nios_setup which contains files related to the qsys setup.
- v) Also the top level file custom_inst.v, the sdc file custom_inst.sdc is included
- 2. Use the command "quartus top" to launch quartus and open the project.
- 3. After quartus launches:
 - i) Go to Tools \rightarrow Programmer.
 - ii) Click on Hardware Setup. A hardware Setup dialog box will open.
 - iii) Select USB Blaster under currently selected hardware. Click on close.
 - iv) Use the Add file tab to navigate to the master/images folder and select the custom_inst.sof file.
 - v) Select the checkbox Program/Configure and Verify.
 - vi) Click on start. This starts the downloading of the .pof file on to the Eval Kit.
- 4. We are using the Nios terminal to display the results. To open the command shell go to:
 Start→All Programs →Altera→Nios II EDS → Nios II Command Shell in Windows or <Nios II EDS install path>/nios2_command_shell.sh in Unix.
- 5. Here, make sure that the reset pin on the Eval Kit is in the "on" position (as the reset is active low). The reset pin is assigned to 5th switch from the left on the SW3 bank of switches. The ON state is written (in very tiny letters on the DIP switch bank) and is the position located when the switch is pushed away from the bottom edge of the PCB.
- 6. In the Nios II command shell type the command: nios2-terminal. The results of the CRC operation will be displayed.

Steps to Recreate the output files:

If you want to edit the project, follow the below procedure:

<u>Hardware</u>

1) Launch Quartus II and open the top.qpf project

File -> Open Project -> top.qpf

2) Compile the project (Processing -> Start Compilation).

3) Program the ./output_files/top.sof to the board using Quartus II Programmer (Tools ->

Programmer - > Add File -> top.sof -> OK -> Start).

<u>Software</u>

1) Open Tools->Nios II Software build tools from Quartus II. This launches the NIOS II Eclipse IDE where you can modify your C Code.

2) Select the workspace for Nios II Eclipse. Then select File->New->Nios II application and BSP from Template

3) In the Window which opens, select the nios_setup.sopcinfo file in your Quartus project folder. The .sopcinfo file contains information about the Qsys system, each module instantiated in the project, and parameter names and values contained in the project.

4) Give the name to your nios project as cust_inst, select hello world small as the project template and click finish. You can see that cust_inst and cust_inst_bsp is created on your workspace.

5) Now, we have to replace the contents of hello_world.c source with our source code. First Remove the helloworld.c file from the project. To do this right click on 'hello_world_small.c' and click delete. (screenshot attached below).

Next the 5 files in the software/src folder.

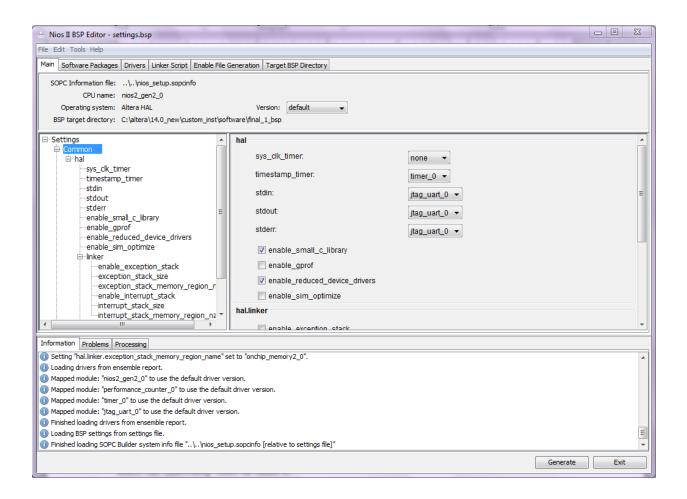
This can be done in 2 ways:

- 1) Directly Copy and Paste the required Files: Directly copy (Ctrl C) the files from the software/src folder. In the Eclipse tool right click on the custom_inst project and click Paste.
- 2) Import the required Files: Right Click on the custom_inst project. Click Import. A new Import popup will open.
 - a. In this double click on General and the File System.
 - b. Select Browse and then browse to the directory software/src and select ok. All the contents of this folder, the 5 .c files are shown in the right. Check the checkboxes for all the 5 files. Click on finish.

Now all the files are present in the folder.

Import	In Implementation /			- O X
File system Import resources from the lo	cal file system.			
From directory: C:\altera\14	.1\bemicro_custom_inst\	oftware\src	•	B <u>r</u> owse
Src		Image: Constraint of the constraint		
Filter <u>Types</u>	All Deselect All			
Into fo <u>l</u> der: new				Bro <u>w</u> se
Options <u>O</u> verwrite existing resource <u>C</u> reate top-level folder <u>A</u> dvanced >>	es without warning			
2			Pi-14	Const
?	< <u>B</u> ack	Next >	<u>F</u> inish	Cancel

6)Since the Qsys contains the timer, this needs to be added here as the timestamp timer. For this right click on the custom_inst_bsp and select NIOS II \rightarrow BSP Editor. A BSP Editor window will open. Here select the timestamp timer as timer 0 and click on generate. After it completes the generation. Click on Exit.



7) Right Click on the custom_inst and Select **Build Project** to build the project. The initial build may take some time.

8) Once the build is finished, to run the project, right click on the project and select Run As -> Run Configurations.

9) Double click on Nios II Hardware, and new configuration opens on the right pane . Make sure you select the project name as custom_inst and .elf file as custom_inst.elf.

10) Select Target Connection Tab . Then Check the following two check boxes Ignore mismatched system ID Ignore mismatched system timestamp

Next Click Apply and Run.

Project I Target Connection	🕸 Debugger	🧤 Source 🕅 🔳	Common		
Connections					
Processors:					
Cable	Device	Device ID	Instance ID	Name	Architecture
USB-BlasterII on localhost	. 10M085(A	. 1	0	nios2 O	Nios2:3
Byte Stream Devices:					
Cable	Device	Device ID	Instance ID	Name	Version
USB-BlasterII on localhost	. 10M085(A	. 1	0	jtaquart 0	1
Disable 'Nios II Console' view					
Quartus Project File name: < Using defau	lt .sopcinfo & .jd	i files extracted f	rom ELF >		
System ID checks					
-					
Ignore mismatched system ID					
Ignore mismatched system timestamp)				
Download					
Download ELF to selected target syst	em				
Start processor					
< L					

11) The following output is observed:

```
📲 Problems 🖉 Tasks 🖳 Console 🔲 Properties 🔚 Nios II Console 🛛
F final_1 Nios II Hardware configuration - cable: USB-BlasterII on localhost [USB-1] device ID: 1 instance ID: 0 name: jtaguart_0
▶ +----+
    | Comparison between software and custom instruction CRC32 |
    +-----+
    System specification
   System clock speed = 50 MHz
   Number of buffer locations = 32
   Size of each buffer = 256 bytes
   Initializing all of the buffers with pseudo-random data
    _____
                            _____
   Initialization completed
   Running the software CRC
   Completed
   Running the optimized software CRC
   Completed
   Running the custom instruction CRC
                _____
   Completed
   Validating the CRC results from all implementations
    _____
   All CRC implementations produced the same results
   Processing time for each implementation
          _____
    Software CRC = 59 ms
    Optimized software CRC = 08 ms
   Custom instruction CRC = 01 ms
    Processing throughput for each implementation
            ------
    Software CRC = 949 Mbps
    Optimized software CRC = 9362 Mbps
    Custom instruction CRC = 1285 Mbps
   Speedup ratio
   Custom instruction CRC vs software CRC = 86
   Custom instruction CRC vs optimized software CRC = 56
   Optimized software CRC vs software CRC= 1
```

The user can modify the number of buffer locations and size of the buffers and run the program to observe the results.

NOTE:

The current project extracted from the .qar already contains the custom component added and working.

However, the steps to add this custom component and generate the Qsys system are shown below for user's reference.

1) Open Qsys from Quartus. From the IP catalogue New component is selected. A new component window will open. The new component's name and display name is entered.

*	Component E	ditor -	CRC_hw	v.tcl*	-								23
F	ile Templates	View											
	Component Type	e 🛛	Files	x	Parameters	x	Signals	X	Interfaces 🛛				- 🗗 🗖
	About Comp	onent T	ype										
	Name:	CRC							1		 		
		CRC]				
		1.0											
	Group:											-	
	Description:												
	Created by:												
	Icon:								-				
	Documentation:	Title			URL								
		+	-							_			

2) Click on Next. The Files Tab is displayed. The folder crc_hw contains the Verilog files, CRC_Component.v and CRC_Custom_Instruction.v for the custom instruction. These files need to be added here.

Next, we have to set the top level entity. For this, double click on the attributes section of the CRC_Custom_Instruction.v file and check the top level file option. This sets the CRC_Custom_Instruction.v file as the top level entity.

Next click on the analyze and synthesize files. This will report compilation errors if any in the .v files.

Component Editor - CR	C_hw.tcl*	searchances 1 beach	-	X
<u>File T</u> emplates <u>V</u> iew				
Component Type 🛛 🛛 🛛	Files 🖾 Parameters 🖾 Signa	ls ⊠ Interfaces ⊠		- 🗗 🗆
About Files				
Synthesis Files				
		eated when a Quartus II synthesis mode ed for this component's parameters and	-	
Output Path	Source File	Type	Attributes	
CRC Component.v		Component.v Verilog HDL	no attributes	
CRC_Custom_Instruction		Custom_In Verilog HDL	Top-level File	
These files will be produced Output Path	when a Verilog simulation model is gene Source File	Type	Attributes	
(No files)				
+ - Copy from S	Synthesis Files			
VHDL Simulation Files				
These files will be produced	when a VHDL simulation model is genera			
Output Path	Source File	Туре	Attributes	
Output Path (No files)	Source Hie	Туре	Attributes	

3) Click on the Next tab. Here the parameters present in the .v files will be displayed.

🚣 Compone	ent Editor - CRC_	hw.tcl*	-	1 0000	-	arrests 1 decar	-	1	X
<u>F</u> ile <u>T</u> empla	ates <u>V</u> iew								
Component	tType 🛛 File	s 🛛 Param	eters	🛛 Signals	x	Interfaces 🛛			- 🗗 🗖
> About F	Parameters								
	Name	Default Value	Edit	Type	Group	Tooltip			
	crc_width	32		integer	Group	roomp			
Parameters:	polynomial_inital	4294967295		logic vec					
Farameters.	polynomial	79764919	~	logic vec			Ξ		
	reflected_input	1	V	integer					
	reflected_output	1	V	integer			-		
	Add Parameter	Remove	Paramet	er					
	Preview the GU	Л							

4) Click on next. The signals tab is displayed. Here, for the 1st signal (the clock) in the interface section select the "New Custom Instruction Slave". After choosing this, the interface section will display the new custom instruction slave as the nios_custom_instruction_slave.

In the interface section select nios_custom_instruction_slavefor all the signals.

In the Signal type select the same option as the signal name.

After selecting all the above the signals tab will have values as shown below.

Component Type 🛛	Files 🔀 Parameters 🔀 Signals 🔀 Inter	faces 🛛		<u> </u>
 About Signals 				
Name	Interface	Signal Type	Width	Direction
clk	nios_custom_instruction_slave	clk	1	input
reset	nios_custom_instruction_slave	reset	1	input
dataa	nios_custom_instruction_slave	dataa	32	input
n	nios_custom_instruction_slave	n	3	input
clk_en	nios_custom_instruction_slave	clk_en	1	input
start	nios_custom_instruction_slave	start	1	input
done	nios_custom_instruction_slave	done	1	output
result	nios_custom_instruction_slave	result	32	output

the	number	of	operand	ls	as 1
	Component Editor - CRC_hw.tcl*	te 1 iteration	Ingeneral I Dealer		23
Eile	<u>T</u> emplates <u>V</u> iew				
	Component Type 🛛 🛛 Files 🖾 Parame	ters 🖾 Signals 🖾	Interfaces 🛛		- 🗗 🗖
	About Interfaces				
	"nios_custom_instruction_slave" (0	Custom Instruction Slave) —			
	Name: nios_custom_instruction_sl	ave Docum	nentation		
	Type: Custom Instruction Slave	•			
	Assignments: Edit				
	🔻 Block Diagram		T Pa	nameters	
				ck cycles: 0	
	nios_custor	n_instruction_slave	Clo	ck cycle type: Variable	
	nios_custom_instruction_slave		Ope	erands: 1	
	clk	clk	- Ac	cess Waveform	
	dataa(31_0)	reset			
	p[20]	dataa n	clk		
	etart	clk_en	cik sta	_en/	
	done	start done	do		
	result[310]	result	n	χ3	/
			null dat	aa 00	
	•	III	▶ res	ult	(RO
•					•
		Add Interface	Remove Interfaces With No Sig	nals	

5) Next Click on Interfaces. Click on Remove Interfaces with no signals. In the Parameters options select

6)Next, Click on finish. Save the changes to CRC_hw.tcl on being prompted.

Now a new component CRC will appear under the new component category in the IP catalogue.

7) Add the other components, such as the Nios, On Chip Memory, Timer, Jtag Uart to the Qsys system and connect the ports. The overall Qsys system looks as below:

1	Use	Connections Name	Description	Export	Clock	Base	End
:	V	⊟ clk_0	Clock Source				
		D− dk_in	Clock Input	clk	exported		
				reset			
:		clk dk	Clock Output	Double-click to ex			
.		dk_res		Double-click to ex	xport		
,	V		_memory2_0 On-Chip Memory (RAM or ROM				
		♦ dk1	Clock Input	Double-click to ex			
1		$\downarrow \phi \phi \rightarrow s1$	Avalon Memory Mapped Slave				0x0000_ffff
2				Double-click to ex	cport [clk1]		
	V	□ nios2_c	jen2_0 Nios II Gen2 Processor (Previe	ew)			
		♦ dk	Clock Input	Double-click to ex			
		♦ → reset	Reset Input	Double-click to ex	cport [clk]		
		data_m	naster Avalon Memory Mapped Maste	er Double-click to ex	cport [clk]		
		instruct	tion_master Avalon Memory Mapped Maste				
			Interrupt Receiver	Double-click to ex	cport [clk]		IRQ 0 IRQ
		debug_	reset_request Reset Output	Double-click to ex	cport [clk]		
		debug_	_mem_slave Avalon Memory Mapped Slave	Double-click to ex	cport [clk]		0x0001_0fff
		custom	_instruction_m Custom Instruction Master	Double-click to ex	(port		
	V	🗆 timer_(0 Interval Timer				
		♦ dk	Clock Input	Double-click to ex	cport clk_0		
		reset	Reset Input	Double-click to ex	cport [clk]		
		s1	Avalon Memory Mapped Slave	Double-click to ex	cport [clk]		0x0001_101f
		irq	Interrupt Sender	Double-click to ex	cport [clk]	_	_
	V	🗆 jtag_ua	art_0 JTAG UART				
	_	◆	Clock Input	Double-click to ex	port clk_0		
		◆ → reset	Reset Input	Double-click to ex	port [dk]		
		avalon	jtag_slave Avalon Memory Mapped Slave	Double-click to ex	port [clk]		0x0001 1027
		irq	Interrupt Sender	Double-click to ex	port [dk]	_	_
	V	⊡ crc_0	crc				
			ustom_instructi Custom Instruction Slave	Double-click to ex	port	Opcode 0	Opcode 0
			- 1		- 1		