

NIOS II Custom Instruction for MAX 10 DE10 - Lite

©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 DE10 – Lite.

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 Nios II Custom Instruction User Guide 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 DE10 are:

1. Extract all the files from the de10_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.
- 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 are at is the a file for executing the built in function for the system instruction.

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. Open the newly created project and compile it in Quartus.
- 3. After compilation completes:
 - 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 output_files folder and select the .sof file.
 - v) Select the checkbox Program/Configure and Verify.
 - vi) Click on start. This starts the downloading of the .sof file on to the DE10.
- 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. 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:

Hardware

1) Launch Quartus II and open the .qpf project

- 2) Compile the project (Processing -> Start Compilation).
- 3) Program the ./output_files/ <project name>.sof to the board using Quartus II

Programmer (Tools -> Programmer -> Add File -> <project name>.sof -> OK -> Start).

Software

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	cación y	
File system Import resources from the local file system.		
From directory: C:\altera\14.1\bemicro_cu	ustom_inst\software\src	▼ B <u>r</u> owse
	♥ € ci_crc.c ♥ € ci_crc.h ♥ € crc_main.c ♥ € crc.c ♥ € crc.h	
Into folder: new Options Qverwrite existing resources without wa Create top-level folder Advanced >>	rning	Bro <u>w</u> se
?	< Back Next >	Einish 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.

Nios II BSP Editor - settings.bsp	and the second se	
File Edit Tools Help		
Main Software Packages Drivers Linker Script Enable File	Generation Target BSP Directory	
SOPC Information file:\\nios_setup.sopcinfo CPU name: nios2_gen2_0 Operating system: Altera HAL BSP target directory: C:\altera\14.0_new\custom_inst\soi	Version: default - tware\final_1_bsp	
Settings Common -hal -sys_cik_timer -timestamp_timer -stdin -stderr -enable_small_c_library -enable_interrupt_stack -exception_stack_size -exception_stack_size -interrupt_stack_size -interrupt_stack_size -interrupt_stack_memory_region_rz /// /// /// /// /// ///	hal sys_clk_timer: none ▼ timestamp_timer: timer_0 ▼ stdin: jtag_uart_0 ▼ stdout: jtag_uart_0 ▼ stderr: jtag_uart_0 ▼ @ enable_small_c_library @ enable_small_c_library @ enable_reduced_device_drivers @ enable_sim_optimize hal.linker	E
Information Problems Processing		
Setting "hallinker, exception_stack_memory_region_name" s Loading drivers from ensemble report. Mapped module: "nios2_gen2_0" to use the default driver version. Mapped module: "timer_0" to use the default driver version. Mapped module: "timer_0" to use the default driver version. Mapped module: "timer_0" to use the default driver version. Mapped module: "timer_0" to use the default driver version. Mapped module: "timer_0" to use the default driver version. Mapped module: "timer_0" to use the default driver version. Mapped module: "timer_0" to use the default driver version. Mapped module: "timer_0" to use the default driver version. Finished loading drivers from ensemble report. Loading BSP settings from settings file. Finished loading SOPC Builder system info file "\.vios_set	et to "onchip_memory2_0". rrsion. It driver version. ion. up.sopcinfo [relative to settings file]"	
	Generate	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.

Connections Processors:					
Cable	Device	Device ID	Instance ID	Name	Architecture
USB-BlasterII on localhost.	10M08S(A	. 1	0	nios2 O	Nios2:3
Byte Stream Devices:					
Cable	Device	Device ID	Instance ID	Name	Version
USB-BlasterII on localhost.	10M08S(A	. 1	0	jtaquart 0	1
Disable 'Nios II Console' view	ault soncinfo & id	files extracted f	rom ELE >		
Disable 'Nios II Console' view Quartus Project File name: < Using def System ID checks	ault .sopcinfo & .jd	i files extracted 1	from ELF >		
Disable 'Nios II Console' view Quartus Project File name: < Using defi System ID checks Ignore mismatched system ID	ault .sopcinfo & .jd	files extracted f	from ELF >		
Disable 'Nios II Console' view Quartus Project File name: < Using defi System ID checks Ignore mismatched system ID Ignore mismatched system timestar	ault .sopcinfo & .jd	files extracted f	from ELF >		
Disable 'Nios II Console' view Quartus Project File name: < Using defi System ID checks Ignore mismatched system ID Ignore mismatched system timestar Download	ault .sopcinfo & .jd	files extracted f	from ELF >		
Disable 'Nios II Console' view Quartus Project File name: < Using defi System ID checks Jgnore mismatched system ID Jgnore mismatched system timestat Download Download ELF to selected target system	ault .sopcinfo & .jd mp ystem	files extracted f	from ELF >		

11) The following output is observed:

```
📲 Problems 🖉 Tasks 📮 Console 🔲 Properties 🔚 Nios II Console 🔀
8
  final_1 Nios II Hardware configuration - cable: USB-BlasterII on localhost [USB-1] device ID: 1 instance ID: 0 name: jtaguart_0
8
   +-----+
    | 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 Typ	e 🖾 🛛 Files	23	Parameters	83	Signals	83	Interfaces 🔀		^
About Comp	onent Type	1 5050			-				
lame:	CRC								
isplay name:	CRC								
ersion:	1.0								
roup:								•	
escription:									
reated by:									
:on:									
ocumentation:	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.

omponent Type 🐰 📕	les 🛞 Parameters 🐰 Signals	s 🖾 Interfaces 🖾		
Abaut Tiles				
About Files				
vnthesis Files				
nese files describe this comp	oonent's implementation, and will be crea	ated when a Quartus II synthesis model i	is generated.	
e parameters and signals f	ound in the top-level module will be use	d for this component's parameters and sig	gnals.	
Output Path	Source File	Туре	Attributes	
RC_Component.v	custom_inst/crc_hw/CRC_0	Component.v Verilog HDL	no attributes	
RC_Custom_Instruction	n.v custom_inst/crc_hw/CRC_C	Custom_In Verilog HDL	Top-level File	
+ - Analyze Syn	thesis Files Create Synthesis Fil	le from Signals		
+ - Analyze Syn	thesis Files Create Synthesis Fil	le from Signals		
+ • Analyze Syn op-level Module: (Analyze f erilog Simulation Files uese files will be produced w	thesis Files Create Synthesis Fil files to select module) 🚽	ie from Signals		
+ - Analyze Syn p-level Module: (Analyze f erilog Simulation Files uese files will be produced w Dutput Path	thesis Files Create Synthesis Fil files to select module) + when a Verilog simulation model is genera Source File	ie from Signals ated. Type	Attributes	
+ Analyze Syn p-level Module: (Analyze t erilog Simulation Files ese files will be produced w utput Path	thesis Files Create Synthesis Fil files to select module) - when a Verilog simulation model is general Source File	le from Signals ated. Туре	Attributes	
+ Analyze Syn p-level Module: (Analyze f rilog Simulation Files ese files will be produced w hutput Path (No files)	thesis Files Create Synthesis Fil files to select module)	ated.	Attributes	
+ Analyze Syn p-level Module: (Analyze f rilog Simulation Files ese files will be produced w utput Path (No files)	thesis Files Create Synthesis Fil files to select module) - when a Verilog simulation model is genera Source File	le from Signals ated. Type	Attributes	
Analyze Syn Analyze Syn p-level Module: (Analyze f rilog Simulation Files ese files will be produced w utput Path (No files) Copy from S	thesis Files Create Synthesis Fil files to select module) + when a Verilog simulation model is genera Source File	ated.	Attributes	
Analyze Syn Analyze Syn p-level Module: (Analyze f rrilog Simulation Files ese files will be produced w utput Path (No files) Copy from S IDL Simulation Files	thesis Files Create Synthesis Fil files to select module) + when a Verilog simulation model is genera Source File	le from Signals ated. Туре	Attributes	
Analyze Syn Analyze Syn p-level Module: (Analyze 1 (Analyze 1 (Analyze 1 (Analyze 1 (No Files) (No Files) (No Files) (DL Simulation Files ese files will be produced w	thesis Files Create Synthesis Fil files to select module)	le from Signals ated. Type	Attributes	
Analyze Syn p-level Module: (Analyze 1 p-level Module: Analyze 1 p-level Module: (Analyze 1 p-level#	thesis Files Create Synthesis Fil files to select module) - when a Verilog simulation model is general Source File ynthesis Files when a VHDL simulation model is generat Source File	ted.	Attributes	
Analyze Syn Analyze Syn o-level Module: (Analyze f rilog Simulation Files ese files will be produced w utput Path (No files) Copy from S DL Simulation Files ese files will be produced w utput Path	thesis Files Create Synthesis Fil files to select module) - when a Verilog simulation model is generat Source File ynthesis Files when a VHDL simulation model is generat Source File	ted.	Attributes	

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

e Tembra	ates <u>v</u> iew								
Componen	tType 🛛 File	es 🖾 🏾 Param	eters	Signals	🕄 Inte	erfaces 🛛			- 🗗
► About F	Parameters								
							_	1	
	Name	Default Value	Edit	Туре	Group	Tooltip			
	crc_width	32	1	integer			-		
arameters:	polynomial_inital	4294967295	V	logic vec					
a carrie cer or	polynomial	79764919	1	logic vec			Ξ		
	reflected_input	1	V	integer				J	
	reflected_output	1	1	integer			-		
		100406 100	D	1		10		1	
	Add Parameter	Remove	Paramet	er					
	Preview the GL	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 원 File	s 🖾 Parameters 🖾 Signals 🗮 Inter	faces 🛛		-
▶ 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	operands	as	1.
🛵 Com	ponent Editor - CRC_hw.tcl*	ter 1 bittersent	ingroups I through		23
<u>File</u> <u>T</u> er	mplates <u>V</u> iew				
Compo	onent Type 🔀 Files 🖾 Parame	ers 🖾 Signals 🖾	Interfaces 🕺		- 🗗 🗖
► Ab	out Interfaces				
	"nios_custom_instruction_slave" (0	ustom Instruction Slave) —			
	Name: nios_custom_instruction_sla	ave Docum	nentation		
	Type: Custom Instruction Slave	•			
	Assignments: Edit				
	- plash process		(Para	mahaur	
	Block Diagram		Clock	cycles: 0	
	nios_custon	_instruction_slave	Clock	cycle type: Variable	
	nios custom instruction slave		Opera	ands: 1	
	<u>clk</u>	sik	Acce	ess Waveform	
	dataa(310)	reset			
	n[20]	n la	cik		
	start	clk_en	CIK_e	en/	
	done	done	done		
	resultation	result	n	3	
			dataa	a(D0	
•		m	resul	t	(RO
•		Ш			•
		Add Interface	Remove Interfaces With No Signa	ls	

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:

Use	Connections	Name	Description	Export	Clock	Base	End	
V	(#	⊟ clk_0	Clock Source					
		□- dk_in D- dk_in_reset < dk dk dk_reset	Clock Input Reset Input Clock Output Reset Output	cik reset Double-click to export Double-click to export	<i>exported</i> clk_0			
			On-Chip Memory (RAM or ROM) Clock Input Avalon Memory Mapped Slave Reset Input	Double-click to export Double-click to export Double-click to export	clk_0 [clk1] [clk1]	# 0x0000_8000	0x0000_ffff	
		inios2_gen2_0 dk reset data_master instruction_master irq debug_reset_request debug_men_slave custom_instruction_m	Nos II Gen2 Processor (Preview) Clock Input Reset Input Avalon Memory Mapped Master Interrupt Receiver Reset Output Avalon Memory Mapped Slave Custom Instruction Master	Double-click to export Double-click to export	clk_0 [clk] [clk] [clk] [clk] [clk] [clk]	# 0x0001_0800	IRQ 0 0x0001_0fff	II
		\exists timer_0 dk \rightarrow reset \rightarrow s1 \forall inc	Interval Timer Clock Input Reset Input Avalon Memory Mapped Slave Interrunt Sender	Double-click to export Double-click to export Double-click to export Double-click to export	clk_0 [clk] [clk]		0x0001_101f	
		→ jtag_uart_0 clk reset avalon_jtag_slave irq	JTAG UART Clock Input Reset Input Avalon Memory Mapped Slave Interrupt Sender	Double-click to export Double-click to export Double-click to export Double-click to export	clk_0 [dk] [dk] [dk]	₽ 0x0001_1020	0x0001_1027	
		←→ crc_0 nios_custom_instructi	crc Custom Instruction Slave	Double-click to export		Opcode 0	Opcode 0	