

Bootable Embedded Systems for the DE0-Nano Board

For Quartus[®] Prime 18.1

1 Introduction

This tutorial explains how the Intel[®] DE0-Nano board can be configured such that, when power is applied, it automatically loads an embedded system including the Nios[®] II processor and runs a boot program. Both the embedded system and Nios II boot program are stored in the non-volatile FPGA configuration device on the DE0-Nano board. This tutorial assumes that the reader has a working knowledge of the C programming language. It also assumes that the reader is familiar with the Intel Quartus[®] Prime and Platform Designer software, as well as the Intel FPGA Monitor Program, and has access to a computer on which this software is installed.

The screen shots in this tutorial were created using Quartus Prime version 18.1, Platform Designer, and Monitor Program software. If other versions of the software are used, the screen images may be slightly different.

Contents:

- Making a bootable embedded system for the DE0-Nano board
- Writing a boot program for the Nios II processor, and storing this program in a memory initialization file
- Changing the contents of the non-volatile FPGA configure device on the DE0-Nano board

2 Introduction

A bootable embedded system for the DEO-Nano board can be created by following these steps:

- 1. Make an embedded system that includes a Nios II processor by using the Quartus Prime and Platform Designer software. Include an on-chip SRAM module in this embedded system.
- 2. Set the address of the reset vector for the Nios II processor to be in the on-chip SRAM module.
- 3. Configure the on-chip SRAM module so that it will be initialized during FPGA programming.
- 4. Create the desired boot program that should be executed by the Nios II processor. Store the compiled boot program in a memory initialization file for the on-chip SRAM module.
- 5. Store the embedded system in the non-volatile FPGA configuration device on the DEO-Nano board.

The above steps are described in detail in the following sections.

3 Making a Bootable Embedded System using Platform Designer

For the purpose of this tutorial we will use the embedded system illustrated in Figure 1. This embedded system is called the *DEO-Nano Basic Computer*, and it is provided with the Intel FPGA Monitor Program software. The source code for this system can be found in the folder where the Intel FPGA Monitor Program is installed on the reader's computer: *<University_Program_Root_Directory>\Computer_Systems\DEO-Nano\DEO-Nano_Basic_Computer*. Note that a different embedded system can also be used with this tutorial, as long as the system includes both a Nios II processor and at least one on-chip SRAM module.



Figure 1. Block diagram of the DE0-Nano Basic Computer.

Qsys - nios_system.qsys* (D:\Bootable_DE0_Nano\nios_system.qsys)							X			
ine Fran Skireni Senerare Tiew Trons Tieth										
🔄 IP Catalog 🛛 🗕 🗗 🗖	12	System	Conter	nts 🖾 Address Map 🖾	Interconnect Requirements 🛛				-	
	System: nios_system Path: SysID									
	H	Line	<u> </u>	Namo	Description	-	Clack	Page	End	TRO
Project		Use	C		Description	E	CIOCK	Dase	Enu	IRQ
New Component	1	V		⊡ Щ System_PLL	System and SDRAM Clocks for DE-seri		exported			
Library =	×	V		표 미교 Nios2	Nios II Processor		System_PL	● 0x0a00_0000	0x0a00_07ff	\leftarrow
Basic Functions				Nios2_Floating_Point	Floating Point Hardware			Opcode 252	Opcode 255	
H-DSP	ll'-	V		田唱 JTAG_to_FPGA_B	JTAG to Avalon Master Bridge		System_PL			
		V		∃ SDRAM	SDRAM Controller		System_PL	● 0x0000_0000	0x01ff_ffff	
E-Low Power	🔷	V		Onchip_SRAM	On-Chip Memory (RAM or ROM)		System_PL	🔒 multiple	multiple	
				① Onchip_SRAM_2	On-Chip Memory (RAM or ROM)		System_PL	â multiple	multiple	
	x.	V			Parallel Port		System_PL		0x1000_001f	
		V		Dip_Switches	Parallel Port		System_PL		0x1000 004f	
New	11			Pushbuttons	Parallel Port		System PL.	● 0x1000 0050	0x1000 005f	≻−fi
	신			Expansion JP1	Parallel Port		System PL.		0x1000 006f	≻-ŭ
🧏 Н 🖾 Devic 🖾 🔚 🗖				Expansion 1P2	Parallel Port		System Pl	▲ 0×1000_0070	0x1000_007f	-17
system_pir_rer_cik	1			Expansion_3P2	Parallel Port		System Pl	A 0x1000_0070	0*1000_0095	
system_pll_ref_reset		×		Expansion_JP3 In	Parallel Port		System_PL.	A 0.1000_0080	0x1000_0081	드립
							System_PL.	■ 0x1000_0090	0x1000_009E	그별
Expansion_JP1				H JIAG_UARI	JIAGUARI		System_PL	• 0x1000_1000	0x1000_1007	믭
Expansion_JP2		V		Interval_Timer	Interval Timer		System_PL	● 0x1000_2000	0x1000_201f	p—0
Expansion_JP3					System ID Peripheral		System_PL	● 0x1000_2020	0x1000_2027	
		•								۴.
I ⊕ Green_LEDs	- De Green_LEDs E Av fit. V S Current filter:									
		_								
I TAG_OART	* =	Message	es δ	3					-	
			-							F
	Тур	e	Pa	m Message						<u> </u>
The Onchin SDAM			2 In	fo Messages					_	÷
4					111					•
0 Errors, 0 Warnings								G	enerate HDL	Finish

Figure 2. The Platform Designer window for the DEO-Nano Basic Computer.

Create a copy of the source code of the DEO-Nano Basic Computer in a folder named *Bootable_DEO_Nano*. As mentioned earlier, this source code can be found in the folder where the Intel FPGA Monitor Program is installed. In the DEO-Nano Basic Computer, the Nios II processor's reset vector is set to the starting address of the SDRAM module. To use the DEO-Nano Basic Computer as a bootable system, the address of the Nios II reset vector has to be modified to use the on-chip SRAM module. To make this change, open the Quartus Prime project for the DEO-Nano Basic Computer, which has the name *DEO_Nano_Basic_Computer.qpf*. In the Quartus Prime software, open the Platform Designer tool, as indicated in Figure 2, and then open the settings for the Nios II processor, shown in Figure 3.

Nios II Processor - Nios2							
Nios II Processor altera_nios2_gen2							
Main Vectors Caches and Memory Interfaces Arithmetic Instructions MMU and MPU Settings JTA							
Reset Vector							
Reset vector memory:	Onchip_SRAM.s2 -						
Reset vector offset:	0x0000000						
Reset vector:	0x08000000						
Exception Vector							
Exception vector memory:	Onchip_SRAM.s2						
Exception vector offset:	0x0000020						
Exception vector:	0x08000020						
Fast TLB Miss Exception Vector							
Fast TLB Miss Exception vector memory:	None						
Fast TLB Miss Exception vector offset:	0x0000000						
Fast TLB Miss Exception vector:	0x0000000						
	Cancel						

Figure 3. Specifying the reset vector address.

Assign the **Reset vector memory** for Nios II to the module called *Onchip_memory_SRAM*, with an *offset* of 0. The address of the reset vector is now set to the starting address of the on-chip SRAM module, which is **0x08000000**.

Figure 3 also shows that the Nios II exceptions vector is set to the on-chip SRAM memory, at the address 0x08000020. This assignment is optional, and does not have to use the same memory module as for the reset vector.

Since the Nios II processor will execute instructions in the on-chip SRAM module, the contents of this memory have to be initialized during FPGA configuration. This is effected by opening the settings windows for the on-chip SRAM module within Platform Designer, as illustrated in Figure 4. Under the Memory initialization heading, select Initialize memory content. Also, select Enable non-default initialization file, and specify the filename *boot_code.hex*. This memory initialization file will be created later in this tutorial.

Mamanubuna		
Туре:	RAM (Writable)	
Dual-port access		
Cincle clock approximation		
Read During Write Mode:	DONT CARE	
Block type:		
Tightly Coupled Memory of	eration require dual port & dual clo	ock sources.
Size		
Data width:	32 🗸	
Total memory size:	8192 by	ytes
Minimize memory block usage (r	nay impact fmax)	
Read latency		
Slave s1 Latency:	1 -	
Slave s2 Latency:	1 -	
ROM/RAM Memory Protection	1	
Reset Request:	Enabled 👻	
ECC Parameter		
Extend the data width to support E	CC bits: Disabled 👻	
Memory initialization		
✓ Initialize memory content		
Enable non-default initialization	file	
Type the filename (e.g: my_r	am.hex) or select the hex file using the file bro	owser button.
User created initialization file:	boot_code.hex	
Enable In-System Memory Cont	ent Editor feature	
Instance ID:	NONE	
	· ·	
User is required to provide Memory will be initialized fi	memory initialization files for memo om boot_code.hex	ry.

Figure 4. Specifying a memory initialization file.

After the Nios II processor and on-chip SRAM module have been configured as needed, a new embedded system can be generated by using Platform Designer. Open the Generation window by selecting Generate > Generate HDL ... in the Platform Designer window and then click on the Generate button. After the embedded system has been successfully generated, the Platform Designer tool can be closed. Now, in Quartus Prime recompile the DE0-Nano Basic Computer project to produce a new FPGA programming file, which is called *DE0_Nano_Basic_Computer.sof*. We should note that we have not yet created the memory initialization file that was specified in Figure 4. Quartus

Prime allows the project to be compiled without including this file—we will create the memory initialization file, called *boot_code.hex*, later in this tutorial, and then compile the Quartus Prime project again.

4 Creating a Boot Program for the Nios[®] II Processor

A boot program provides the code that the Nios II processor executes when power is applied, or a reset is performed. For this tutorial we use a small example of a boot program that performs the simple task of scrolling a light back and forth across the green LEDs on the DEO-Nano board. This program is provided only as an example to illustrate the steps involved—in a real application a boot program would perform a more useful task. In many practical applications it may be desirable on power-up to run a Nios II program that is too large to fit into the provided on-chip SRAM module. In such cases, the boot program would be designed to load a larger program from a non-volatile storage location, such as the EEPROM chip on the DEO-Nano board, into the SDRAM chip.

The example boot program for this tutorial is shown in Figure 5. It uses memory mapped I/O to load a pattern into the parallel port of the DEO-Nano Basic Computer that is connected to the green LEDs on the DEO-Nano board. The pattern is repeatedly displayed on the LEDs after being shifted in the right or left direction. The frequency of shifting and displaying the pattern is controlled by using the interval timer in the DEO-Nano Basic Computer.

Type the code in Figure 5 into a file with the name *boot_code.c*.

```
/* This program sweeps a green light back and forth on the LEDG lights */
enum DIR { LEFT, RIGHT };
void sweep(int *, enum DIR *);
int main(void) {
    volatile int * LEDG_ptr = (int *)0x10000010; // green LED address
    volatile int * timer_ptr = (int *)0x10002000; // interval timer address
            LEDG_bits = 1; // pattern for the green lights
    int
    enum DIR shift dir = LEFT; // pattern shifting direction
    /* set the interval timer period */
    int counter = 0x190000; // 1/(50 MHz) x 0x190000 = 33 msec
    *(timer_ptr + 2) = (counter & 0xFFFF);
    *(timer_ptr + 3) = (counter >> 16) & 0xFFFF;
   *(timer_ptr + 1) = 0x6; // START = 1, CONT = 1, ITO = 0
   while (1) {
        *LEDG_ptr = LEDG_bits; // write to the green lights
        sweep(&LEDG_bits, &shift_dir); // shift the pattern left or right
       while ((*timer_ptr & 0x1) == 0)
                // wait for timeout
           ;
        *timer_ptr = 0; // reset the timeout bit
    }
}
/* shift the pattern shown on the LEDs */
void sweep(int * pattern, enum DIR * dir) {
    if (*dir == LEFT)
       if (*pattern & 0x80)
           *dir = RIGHT;
       else
           *pattern = *pattern << 1;</pre>
    else if (*pattern & 0x01)
       *dir = LEFT;
    else
       *pattern = *pattern >> 1;
}
```

Figure 5. The boot program.

To compile and test the boot program, ensure that a DEO-Nano board is plugged into your computer. Then, open the Intel FPGA Monitor Program software. In the Monitor Program, create a new project by using the New Project Wizard. Give the project the name *boot_code*. In the Select a system screen of the New Project Wizard, choose Custom System, as illustrated in Figure 6, and browse to select the embedded system called *nios_system.qsys* that we created using the Platform Designer tool in the *Bootable_DEO_Nano* folder. Also browse to select the Quartus Prime programming (SOF) file that we made in the previous step of this tutorial, named *DEO_Nano_Basic_Computer.sof*.

New Project Wizard	x
Specify a system	
Select a system	
<custom system=""> Documentation</custom>	
Specify a system by selecting a system description (SOPCInfo) file, and optional Quartus II programming (SOF) and Quartus II JTAG debugging information (JDI) files.	
System details System description file (SOPCInfo):	
D:/Bootable_DE0_Nano/verilog/nios_system.sopcinfo Browse	
Quartus II programming (SOF) file:	
D:/Bootable_DE0_Nano/verilog/DE0_Nano_Basic_Computer.sof Browse	
The SOF file represents the FPGA programming file for the hardware system. If it is specified here, then the Monitor Program can be used to download this programming file onto the board. Otherwise, the system will need to be downloaded using some other method (for example, by using Quartus II).	
< <u>Back</u> <u>N</u> ext > <u>Finish</u> <u>Canc</u>	:el

Figure 6. Selecting the bootable embedded system.

Click Next in the New Project Wizard, and in the Specify a program type screen select *C Program*. Next, in the Specify program details screen click the Add button and choose the *boot_code.c* file. Accept the default settings

in the Specify system parameters screen, including the specification of the *USB-Blaster* cable that is connected to your DE0-Nano board. Finally, in the Specify program memory settings screen, shown in Figure 7, set the *.text* section of the Nios II program to the on-chip SRAM module.

New Project Wizard	men	nory settings	X		
Memory options	ction n	ames and their start and e	nd addresses. These sections will be used by		
the linker to place code by the linker, the names	and da s must r	ta at the specified address match those identified by	the assembler directives, such as .text.		
Linker Section Presets: Basic 🗸					
Section Name		Memory Device	Address Range		
.text	Or	nchip_SRAM	0x08000000 - 0x08001FFF		

Figure 7. Specifying the memory module for the text and data sections.

Compile the *boot_code* project in the Monitor Program and download it into your DE0-Nano board. Run the program and observe the scrolling pattern on the green lights.

As a result of compiling the *boot_code* project an executable file named *boot_code.elf* was created, and then down-loaded into the on-chip SRAM module by the Monitor Program. To use this executable file as a boot program, we need to store it in the memory initialization file that we specified in Figure 4, and then include this file in the data that is stored in the non-volatile FPGA configuration device on the DE0-Nano board. We need to first convert the *elf* file it into a different format, as described below.

4.1 Using a Memory Initialization File

To make a memory initialization file, we need to convert the *boot_code.elf* file into a format known as *Intel HEX* format. To perform the conversion select the Monitor Program command Actions > Convert Program to Hex File. This command will create a file in Intel HEX format with the name *nios_system_Onchip_SRAM.hex*. Rename the file *boot_code.hex*.

Make sure that a copy of the *boot_code.hex* file is present in the *Bootable_DE0_Nano* folder where the Quartus Prime project for this tutorial is stored. Then, recompile the project in Quartus Prime to make a new FPGA programming file, which is named *DE0_Nano_Basic_Computer.sof*. Since we specified, in Figure 4, that *boot_code.hex* should be used to initialize the on-chip SRAM module, then the *SOF* file produced by Quartus Prime will now contain this information. We can permanently store the contents of this *SOF* file into the non-volatile FPGA configuration device on the DE0-Nano board as described in the following section.

5 Programming the Non-volatile FPGA Configuration Device

To program the non-volatile FPGA configuration device on the DEO-Nano board we need to use a method called *JTAG* Indirect Configuration*. To create the programming information needed for this method, in the Quartus Prime software select the command File > Convert Programming Files. In the window shown in Figure 8, click on the pull-down menu next to Programming file type and select JTAG Indirect Configuration File (.jic).

e <u>T</u> ools <u>W</u> indow				Search altera.com
peofy the input files te ou can also import inp iture use. Conversion setup files Open Output programming file type Options/Boot info File name: Advanced	convert and the ty ut file information fm is Conversion Setup D le Programmer Obje Raw Binary File Programmer Obje Raw Pinary File Programmer Obje Raw Programmer Obje Raw Programmer Obje Raw Programmer Obje Programmer Obje Raw Programmer Obje	ec of programming file to gene om other files and save the co ata bt-Format) Output File for SRAI ct File (pof) (ttf) (ttf) g Data File (rpd) figuration File (co) RAM Object File (porsf) or Partial Reconfiguration (rb or Partial Reconfiguration)	rate. nversion setup information of Save Conversion II (.hexout) f))	Setup
Input files to convert	Create config	data RPD (Generate output_fil jection File (Generate output_	e_auto.rpd) file.fif)	
File/Da	ta area	Properties	Start Address	Add Hex Data
Options			0x00010000	Add Sof Page
SOF Data		Page 0	<auto></auto>	Add Sof Page
oor bata				

Figure 8. Choosing the programming file type.

As illustrated in Figure 9, make the following settings. Under Configuration device select either *EPCS64* (for most DE0-Nano boards) or *EPCS16* (for older DE0-Nano boards). In the File name box specify a meaningful name for the file that will be generated, such as *Bootable_DE0_Nano.jic*. Next, as indicated in the figure, in the Input

files to convert area click to highlight the line SOF Data, and and click Add File to browse to select the file *DE0_Nano_Basic_Computer.sof*.

becify the input files to ou can also import input	convert and the type o t file information from o	of programming file to gen other files and save the c	erate. onversion setup	information creat	ted here for
Conversion setup files					
Open C	onversion Setup Data.		Save	e Conversion Setu	ıp
Output programming file	,				
Programming file type:	JTAG Indirect Config	uration File (.jic)			
Options/Boot info	Configuration device:	EPCS64	Mode:	Active Serial	
File name: Bootable_DE0_Nano.jic					
Advanced	Remote/Local update	difference file: NONE			
	Create Memory Ma	ap File (Generate Bootabl	e_DE0_Nano.ma	ap)	
	Create CvP files (Generate Bootable_DE0_	Nano.periph.jic a	and Bootable_DE0	_Nano.core.rbf)
	Create config data	a RPD (Generate Bootable	e_DE0_Nano_au	to.rpd)	
	Create Fault Inject	tion File (Generate Bootat	ole_DE0_Nano.fi	f)	
Input files to convert					
input mes to convert		Properties	St	art Address	Add Hex Data
File/Dat	a area	riopenies			

Figure 9. Additional settings.

Next, click to highlight the line Flash Loader and then click Add Device, as shown in Figure 10. Select the Cyclone[®] IV E device on the DE0-Nano board, which is called EP4CE22, as depicted Figure 11.

		<u>W</u> indow				Searc	ch altera.com
ou ci ture	an also im use.	nport inpu	convert and the type t file information from	of programming file to gene other files and save the co	ate. iversion setup	information create	ed here for
Conv	version se	etup files Open (Conversion Setup Data	a	Sav	e Conversion Setu	ıp
Outp	out progra	mming fil	9				
Prog	gramming	file type:	JTAG Indirect Config	uration File (.jic)			•
Ор	otions/Boo	t info	Configuration device	EPCS64	Mode:	Active Serial	•
File	name:		Bootable_DE0_Nand	.jic			
	Advance	ed	Remote/Local update	difference file: NONE			~
			Create Memory M	ap File (Generate Bootable	DE0 Nano.ma	D)	
			Create CvP files	Generate Bootable DE0 N	no.periph.iic a	nd Bootable DE0	Nano.core.rbf)
			Create config dat	a RPD (Generate Bootable	DEO Nano aut	to.rod)	
			Create Fault Inject	tion File (Generate Bootable	DE0 Nano.fif)	
Input	t files to c	onvert					
		Eile/Det		Properties	Ch	d Address	Add Hay Data
	Flash I o	File/Dat ader	a area	Properties	Sta	IT Address	
	005.0-4						
⊿	SOF Data	a		Page_0	<auto< td=""><td>></td><td>Add Sof Page</td></auto<>	>	Add Sof Page
4	DE0_	a _Nano_B	asic_Computer.sof	Page_0 EP4CE22F17	<auto< td=""><td>></td><td>Add Sof Page</td></auto<>	>	Add Sof Page
4	DE0	a _Nano_B	asic_Computer.sof	Page_0 EP4CE22F17	<auto< td=""><td>></td><td>Add Sof Page Add File Remove</td></auto<>	>	Add Sof Page Add File Remove
1	DE0	a _Nano_B	asic_Computer.sof	Page_0 EP4CE22F17	<auto< td=""><td>></td><td>Add Sof Page Add File Remove</td></auto<>	>	Add Sof Page Add File Remove
4	DE0_	a _Nano_B	asic_Computer.sof	Page_0 EP4CE22F17	<auto< td=""><td>></td><td>Add Sof Page Add File Remove Up</td></auto<>	>	Add Sof Page Add File Remove Up
4	DE0	a _Nano_B	asic_Computer.sof	Page_0 EP4CE22F17	<auto< td=""><td>></td><td>Add Sof Page Add File Remove Up Down</td></auto<>	>	Add Sof Page Add File Remove Up Down
4	DE0	a _Nano_B	asic_Computer.sof	Page_0 EP4CE22F17	<auto< td=""><td>></td><td>Add Sof Page Add File Remove Up Down Properties</td></auto<>	>	Add Sof Page Add File Remove Up Down Properties

Figure 10. Adding the JTAG Indirect Configuration device.

Select Devices	X
Device family	Device name
APEX20K ▲ Arria 10 ▲ Arria 10 ▲ Arria 110X ▲ Cyclone 110C ➡ Cyclone 111C ➡ HardCopy 111 ➡ HardCopy 111 ➡ HardCopy 111 ➡ MAX 100 ➡	EP4CE10 New EP4CE15 Import EP4CE2 Export EP4CE30 Edt EP4CE5 Remove EP4CE6 Uncheck All
	OK Cancel

Figure 11. Choosing the EP4CE22 device.

Finally, under SOF Data click to select the filename *DE0_Nano_Basic_Computer.sof* that we added previously. Click Properties on the righthand side of the screen in Figure 10 to open the window in Figure 12, and click to select Compression.

SOF File Prop	erties
Compression	
ОК	Cancel
]

Figure 12. The Properties window.

To produce the *Bootable_DE0_Nano.jic* programming file, click the **Generate** button on the bottom of the screen in Figure 10.

5.1 Downloading the JIC file into the DE0-Nano Board

To load the generated JIC file into the non-volatile FPGA configuration device on the DE0-Nano board, open the Quartus Prime Programmer. Ensure that the Intel FPGA Monitor Program software is closed (or disconnected from the DE0-Nano board), because the Quartus Prime Programmer cannot communicate with the DE0-Nano board if it is already connected to the Monitor Program.

In the window shown in Figure 13 make sure that the USB-Blaster connected to your DE0-Nano board is shown in the Hardware Setup box. If any programming file, such as the *DE0_Nano_Basic_Computer.sof*, is listed in the Programmer window, then click to select this file and then click on the Delete button. Next, click the Add File button and browse to choose the *Bootable_DE0_Nano.jic* file. Check the Program/Configure, Verify, and Blank-Check selections, as shown in the figure, and then click on the Start button.

The Quartus Prime Programmer will download the JIC file into the non-volatile FPGA configuration device. Once the programming task is completed, power-cycle the DE0-Nano board and verify that it automatically loads and executes the boot program.



Figure 13. The Programmer window.

Copyright © Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Avalon, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel 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 Intel. Intel 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.

*Other names and brands may be claimed as the property of others.