Bootable Embedded Systems for the DE0-Nano Board

For Quartus Prime 15.1

1 Introduction

This tutorial explains how the Altera 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 Altera Quartus II and Qsys software, as well as the Altera Monitor Program, and has access to a computer on which this software is installed.

The screen shots in this tutorial were created using version 15.0 of the Quartus II, Qsys, 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 DE0-Nano board can be created by following these steps:

1. Make an embedded system that includes a Nios II processor by using the Quartus II and Qsys 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 DE0-Nano board.

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

3 Making a Bootable Embedded System using Qsys

For the purpose of this tutorial we will use the embedded system illustrated in Figure 1. This embedded system is called the *DE0-Nano Basic Computer*, and it is provided with the Altera Monitor Program software. The source code for this system can be found in the folder where the Altera Monitor Program is installed on the reader's computer: *<University_Program_Root_Directory> \Computer_Systems\DE0-Nano\DE0-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.

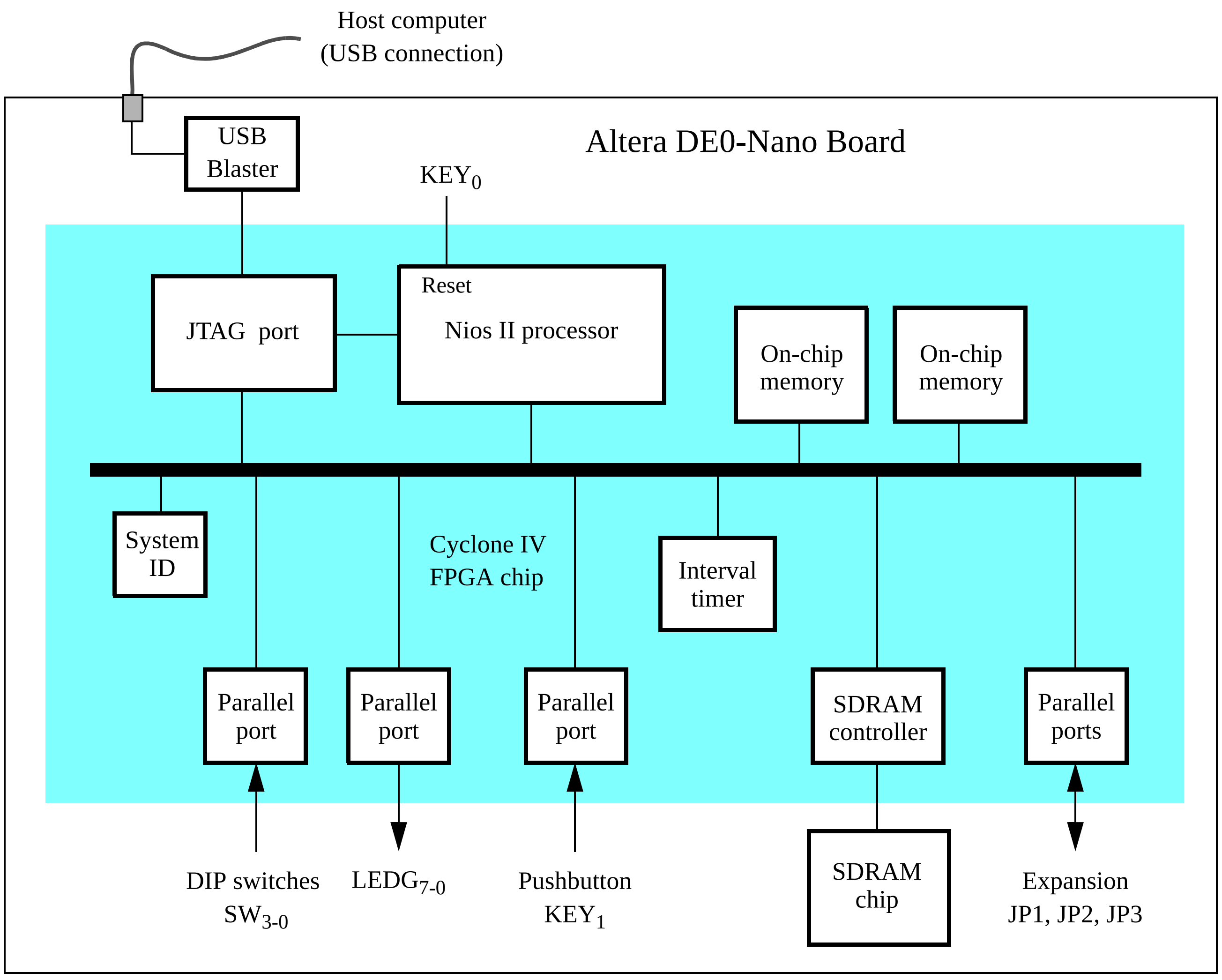
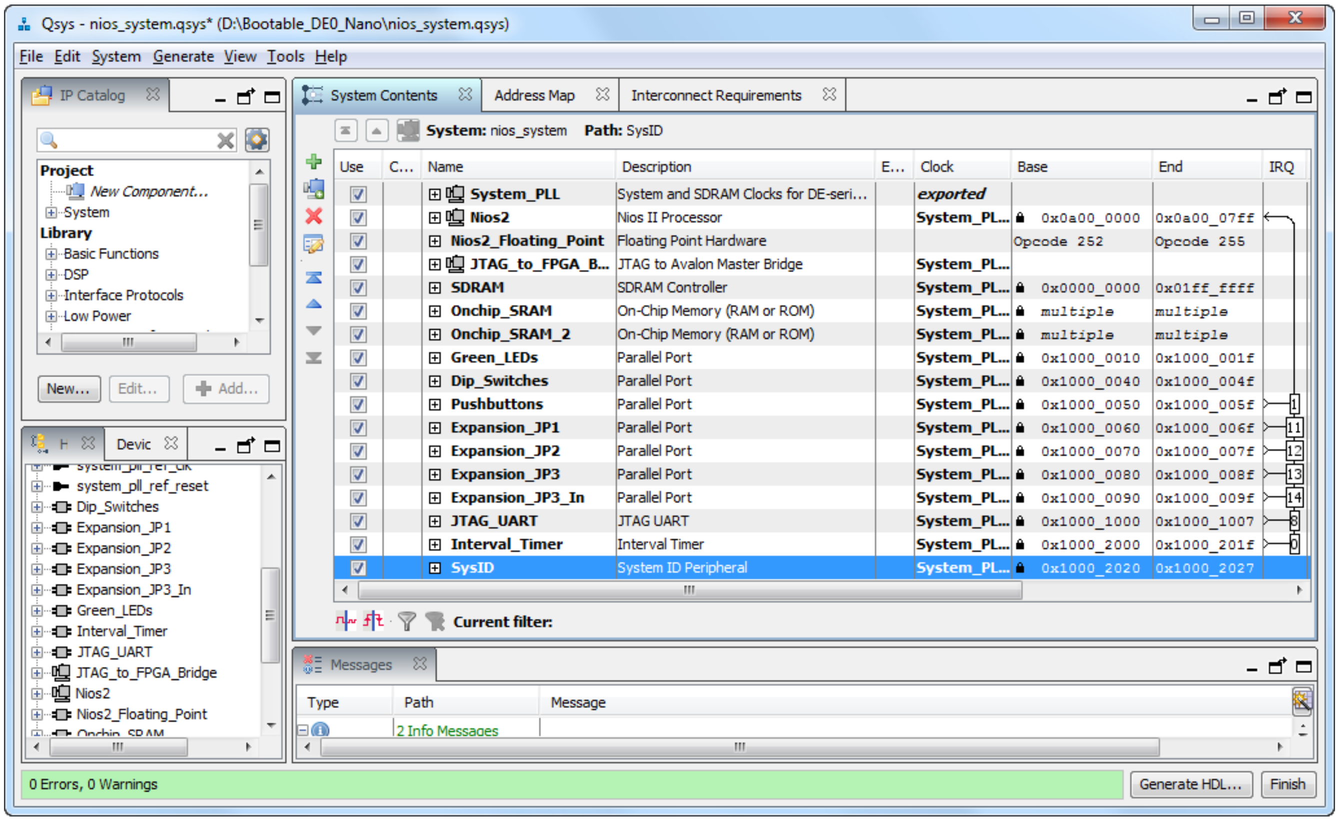


Figure 1. Block diagram of the DE0-Nano Basic Computer.


Qsys - nios_system.qsys* (D:\Bootable_DE0_Nano\nios_system.qsys)
File Edit System Generate View Tools Help
IP Catalog
System Contents
Address Map
Interconnect Requirements
System: nios_system Path: SysID
Current filter:
Messages
Generate HDL...
Finish

Use	C...	Name	Description	E...	Clock	Base	End	IRQ
✓		System_PLL	System and SDRAM Clocks for DE-seri...		*exported*			
✓		Nios2	Nios II Processor		System_PL...	0x0a00_0000	0x0a00_07ff	
✓		Nios2_Floating_Point	Floating Point Hardware			Opcode 252	Opcode 255	
✓		JTAG_to_FPGA_B...	JTAG to Avalon Master Bridge		System_PL...			
✓		SDRAM	SDRAM Controller		System_PL...	0x0000_0000	0x01ff_ffff	
✓		Onchip_SRAM	On-Chip Memory (RAM or ROM)		System_PL...	multiple	multiple	
✓		Onchip_SRAM_2	On-Chip Memory (RAM or ROM)		System_PL...	multiple	multiple	
✓		Green_LEDs	Parallel Port		System_PL...	0x1000_0010	0x1000_001f	
✓		Dip_Switches	Parallel Port		System_PL...	0x1000_0040	0x1000_004f	
✓		Pushbuttons	Parallel Port		System_PL...	0x1000_0050	0x1000_005f	1
✓		Expansion_JP1	Parallel Port		System_PL...	0x1000_0060	0x1000_006f	11
✓		Expansion_JP2	Parallel Port		System_PL...	0x1000_0070	0x1000_007f	12
✓		Expansion_JP3	Parallel Port		System_PL...	0x1000_0080	0x1000_008f	13
✓		Expansion_JP3_In	Parallel Port		System_PL...	0x1000_0090	0x1000_009f	14
✓		JTAG_UART	JTAG UART		System_PL...	0x1000_1000	0x1000_1007	8
✓		Interval_Timer	Interval Timer		System_PL...	0x1000_2000	0x1000_201f	0
✓		SysID	System ID Peripheral		System_PL...	0x1000_2020	0x1000_2027	

Project
New Component...
System
Library
Basic Functions
DSP
Interface Protocols
Low Power

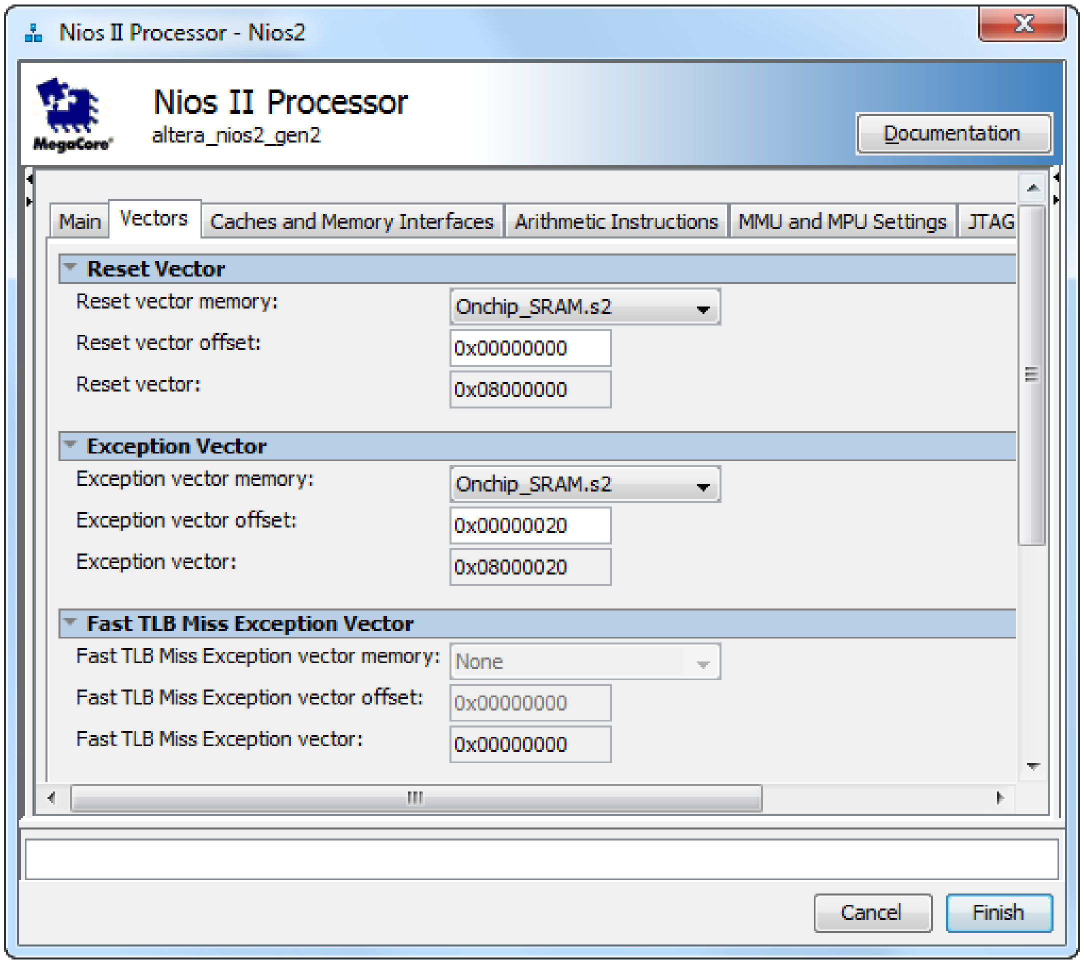
system_pll_ref_reset
Dip_Switches
Expansion_JP1
Expansion_JP2
Expansion_JP3
Expansion_JP3_In
Green_LEDs
Interval_Timer
JTAG_UART
JTAG_to_FPGA_Bridge
Nios2
Nios2_Floating_Point

Type	Path	Message
	2 Info Messages	

0 Errors, 0 Warnings

Figure 2. The Qsys window for the DE0-Nano Basic Computer.

Create a copy of the source code of the DE0-Nano Basic Computer in a folder named *Bootable_DE0_Nano*. As mentioned earlier, this source code can be found in the folder where the Altera Monitor Program is installed. In the DE0-Nano Basic Computer, the Nios II processor's reset vector is set to the starting address of the SDRAM module. To use the DE0-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 II project for the DE0-Nano Basic Computer, which has the name *DE0_Nano_Basic_Computer.qpf*. In the Quartus II software, open the Qsys tool, as indicated in Figure 2, and then open the settings for the Nios II processor, shown in Figure 3.



Reset Vector

Reset vector memory: Onchip_SRAM.s2
Reset vector offset: 0x00000000
Reset vector: 0x08000000

Exception Vector

Exception vector memory: Onchip_SRAM.s2
Exception vector offset: 0x00000020
Exception vector: 0x08000020

Fast TLB Miss Exception Vector

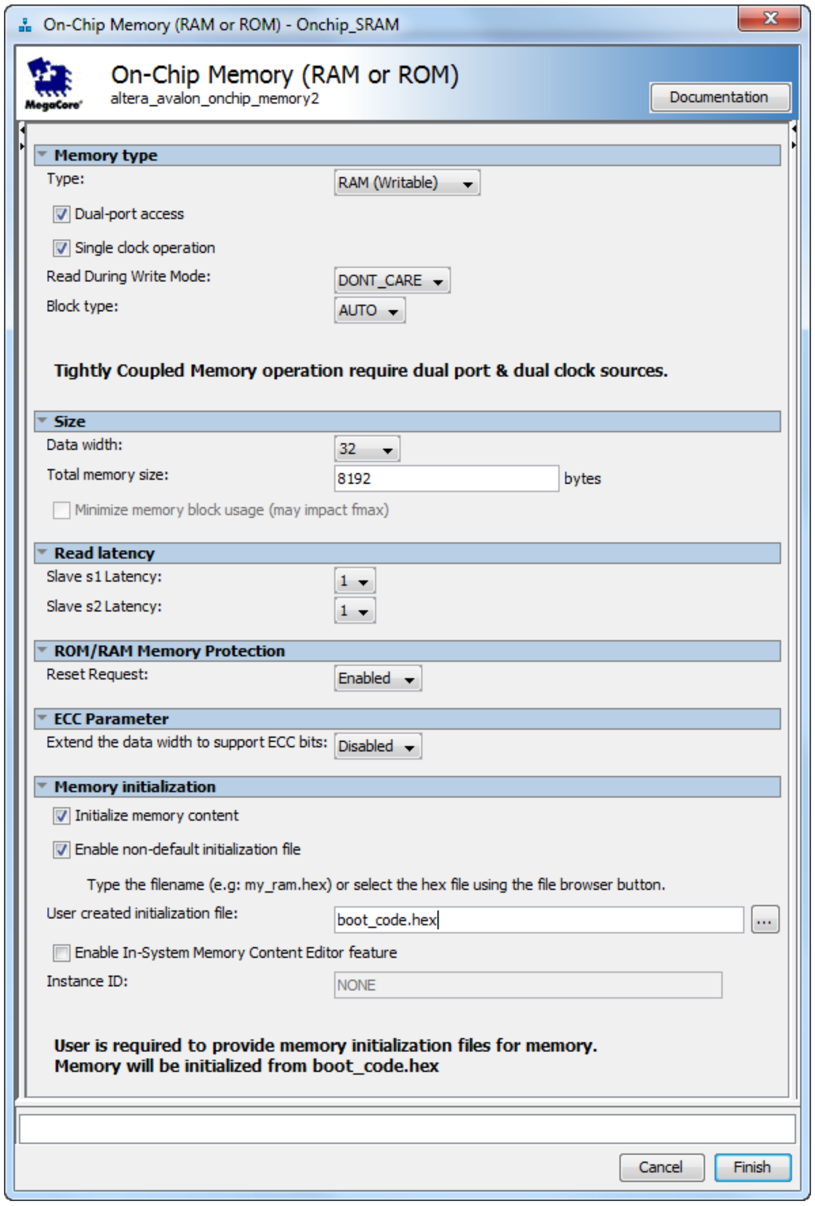
Fast TLB Miss Exception vector memory: None
Fast TLB Miss Exception vector offset: 0x00000000
Fast TLB Miss Exception vector: 0x00000000

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 Qsys, 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.


On-Chip Memory (RAM or ROM) - Onchip_SRAM

On-Chip Memory (RAM or ROM)
altera_avalon_onchip_memory2

Memory type
Type: RAM (Writable)
[✓] Dual-port access
[✓] Single clock operation
Read During Write Mode: DONT_CARE
Block type: AUTO

Tightly Coupled Memory operation require dual port & dual clock sources.

Size
Data width: 32
Total memory size: 8192 bytes
[ ] Minimize memory block usage (may impact fmax)

Read latency
Slave s1 Latency: 1
Slave s2 Latency: 1

ROM/RAM Memory Protection
Reset Request: Enabled

ECC Parameter
Extend the data width to support ECC bits: Disabled

Memory initialization
[✓] Initialize memory content
[✓] Enable non-default initialization file
Type the filename (e.g: my_ram.hex) or select the hex file using the file browser button.
User created initialization file: boot_code.hex
[ ] Enable In-System Memory Content Editor feature
Instance ID: NONE

User is required to provide memory initialization files for memory.
Memory will be initialized from boot_code.hex

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 Qsys. Open the Generation window by selecting Generate > Generate HDL ... in the Qsys window and then click on the Generate button. After the embedded system has been successfully generated, the Qsys tool can be closed. Now, in Quartus II 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 II 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 II 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 DE0-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 DE0-Nano board, into the SDRAM chip.

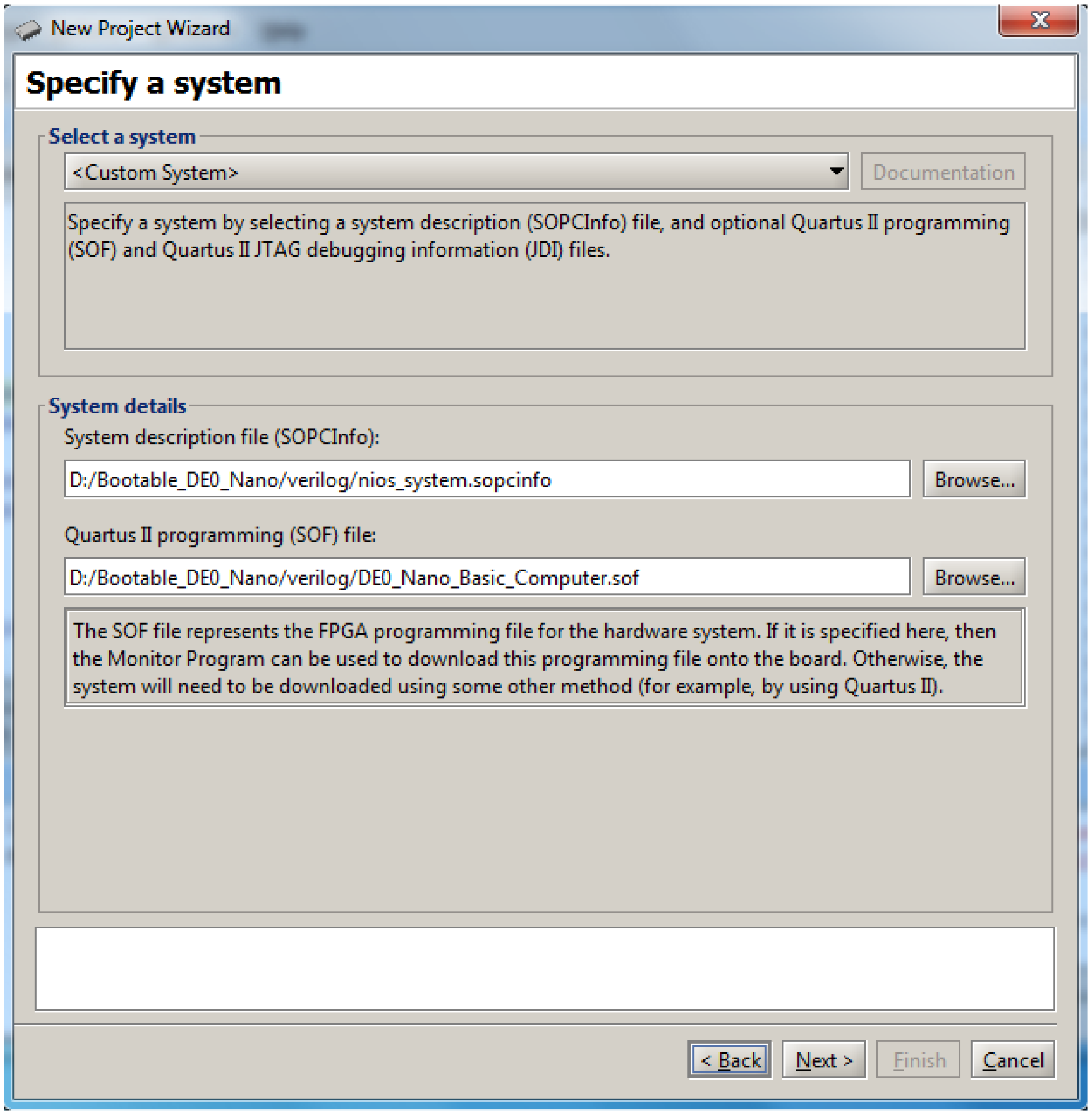
The example boot program for this tutorial is shown in Figure 5. It uses memory mapped I/O to load a pattern into the parallel port of the DE0-Nano Basic Computer that is connected to the green LEDs on the DE0-Nano board. The pattern is repeatedly displayed on the LEDs after being shifted in the right or left direction. The frequency of shifting and displaying the pattern is controlled by using the interval timer in the DE0-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

    int LEDG_bits = 1;                                  // pattern for the green lights
    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;
    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 DE0-Nano board is plugged into your computer. Then, open the Altera 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 Qsys tool in the *Bootable_DE0_Nano* folder. Also browse to select the Quartus II programming (SOF) file that we made in the previous step of this tutorial, named *DE0_Nano_Basic_Computer.sof*.



Specify a system by selecting a system description (SOPCInfo) file, and optional Quartus II programming (SOF) and Quartus II JTAG debugging information (JDI) files.

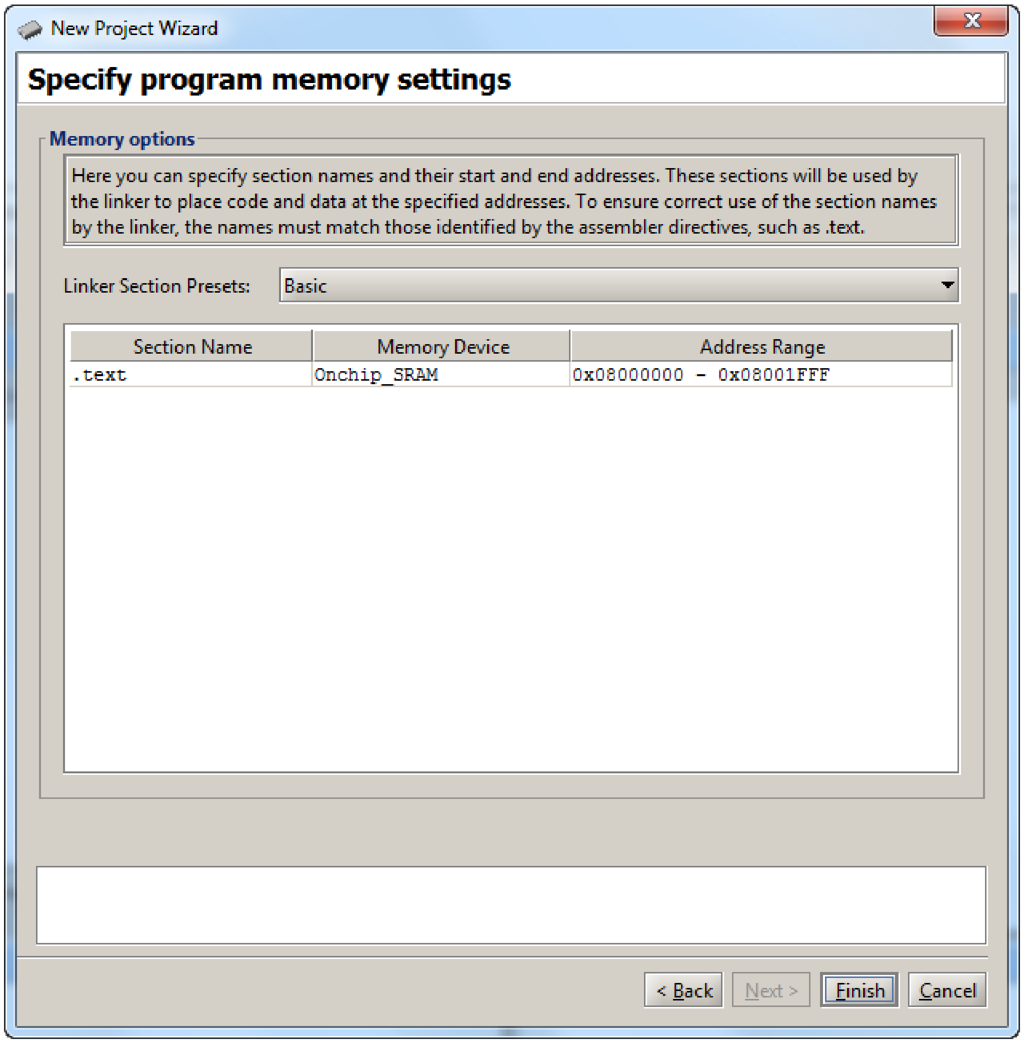
System description file (SOPCInfo): D:/Bootable_DE0_Nano/verilog/nios_system.sopcinfo

Quartus II programming (SOF) file: D:/Bootable_DE0_Nano/verilog/DE0_Nano_Basic_Computer.sof

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).

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

Specify program memory settings

Memory options

Here you can specify section names and their start and end addresses. These sections will be used by the linker to place code and data at the specified addresses. To ensure correct use of the section names by the linker, the names must match those identified by the assembler directives, such as .text.

Linker Section Presets: Basic

Section Name	Memory Device	Address Range
.text	Onchip_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 downloaded 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.

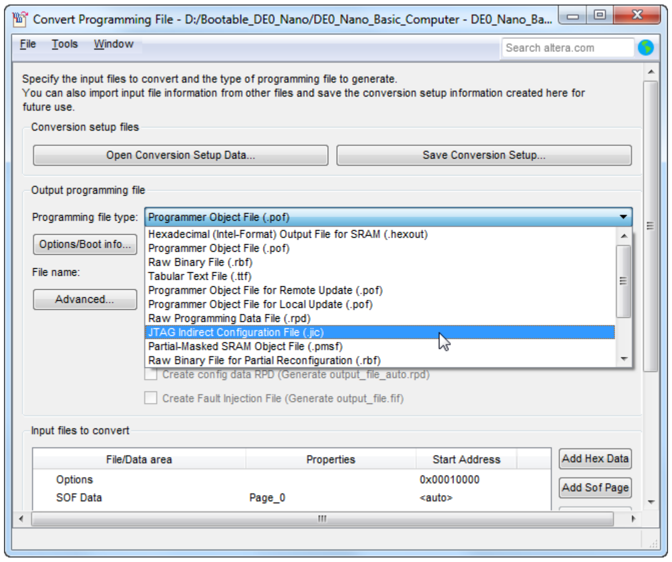
4.1 Using a Memory Initialization File

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 *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 II project for this tutorial is stored. Then, recompile the project in Quartus II 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 II 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 DE0-Nano board we need to use a method called *JTAG Indirect Configuration*. To create the programming information needed for this method, in the Quartus II 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).


Convert Programming File - D:/Bootable_DE0_Nano/DE0_Nano_Basic_Computer - DE0_Nano_Ba...

Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Programming file type: Programmer Object File (.pof)

- Hexadecimal (Intel-Format) Output File for SRAM (.hexout)
- Programmer Object File (.pof)
- Raw Binary File (.rbf)
- Tabular Text File (.ttf)
- Programmer Object File for Remote Update (.pof)
- Programmer Object File for Local Update (.pof)
- Raw Programming Data File (.rpd)
- JTAG Indirect Configuration File (.jic)
- Partial-Masked SRAM Object File (.pmsf)
- Raw Binary File for Partial Reconfiguration (.rbf)

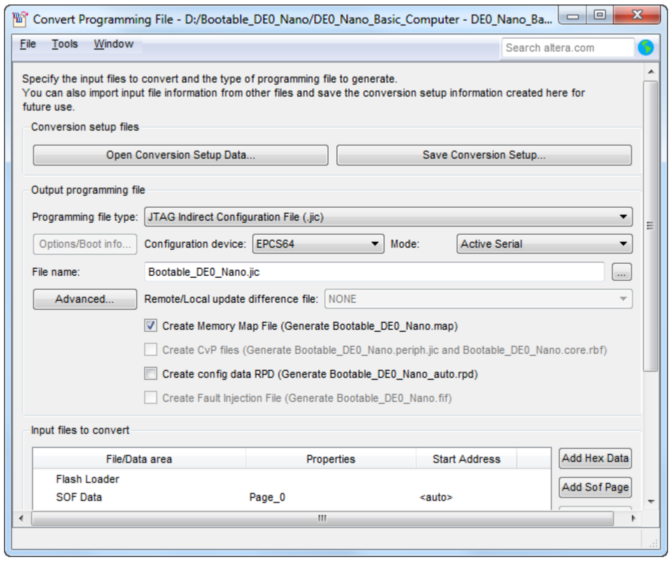
Create config data RPD (Generate output_file_auto.rpd)
Create Fault Injection File (Generate output_file.fif)

Input files to convert

File/Data area	Properties	Start Address
Options		0x00010000
SOF Data	Page_0	<auto>

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*.


Convert Programming File - D:/Bootable_DE0_Nano/DE0_Nano_Basic_Computer - DE0_Nano_Ba...
File
Tools
Window
Search altera.com
Open Conversion Setup Data...
Save Conversion Setup...
Options/Boot info...
Advanced...
Add Hex Data
Add Sof Page

Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Conversion setup files

Output programming file

Programming file type: JTAG Indirect Configuration File (.jic)

Configuration device: EPCS64 Mode: Active Serial

File name: Bootable_DE0_Nano.jic

Remote/Local update difference file: NONE

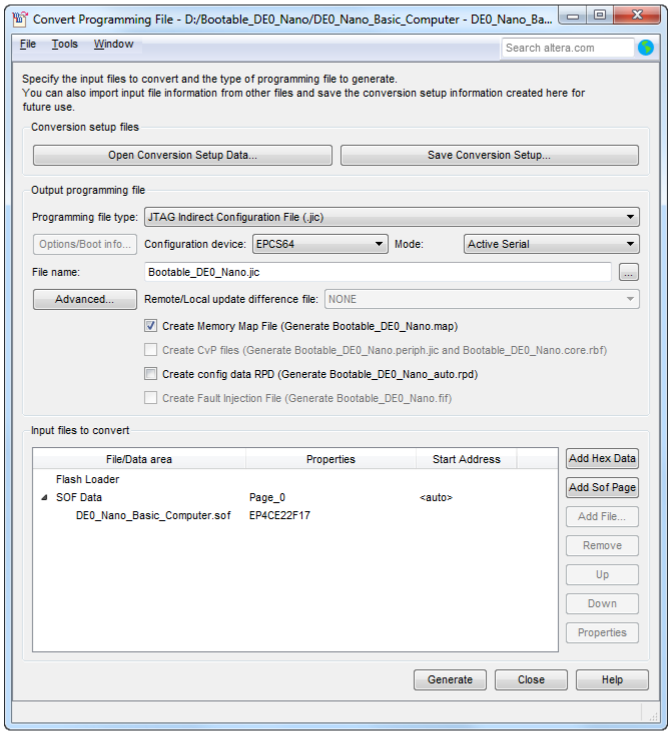
- [✓] Create Memory Map File (Generate Bootable_DE0_Nano.map)
- [ ] Create CvP files (Generate Bootable_DE0_Nano.periph.jic and Bootable_DE0_Nano.core.rbf)
- [ ] Create config data RPD (Generate Bootable_DE0_Nano_auto.rpd)
- [ ] Create Fault Injection File (Generate Bootable_DE0_Nano.fif)

Input files to convert

File/Data area	Properties	Start Address
Flash Loader		
SOF Data	Page_0	<auto>

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.



Specify the input files to convert and the type of programming file to generate.
You can also import input file information from other files and save the conversion setup information created here for future use.

Conversion setup files

Output programming file

Programming file type: JTAG Indirect Configuration File (.jic)

Configuration device: EPCS64 Mode: Active Serial

File name: Bootable_DE0_Nano.jic

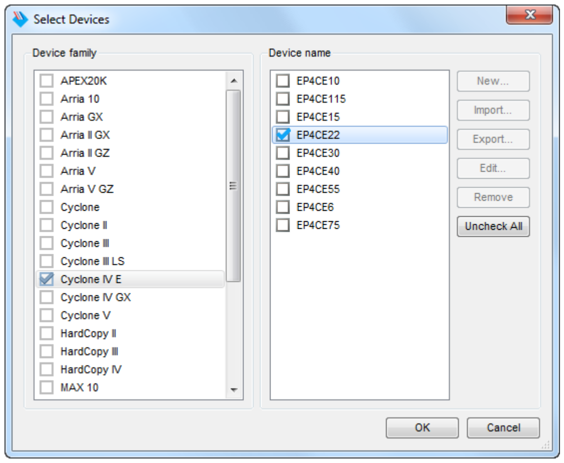
Remote/Local update difference file: NONE

- [✓] Create Memory Map File (Generate Bootable_DE0_Nano.map)
- [ ] Create CvP files (Generate Bootable_DE0_Nano.periph.jic and Bootable_DE0_Nano.core.rbf)
- [ ] Create config data RPD (Generate Bootable_DE0_Nano_auto.rpd)
- [ ] Create Fault Injection File (Generate Bootable_DE0_Nano.fif)

Input files to convert

File/Data area	Properties	Start Address
Flash Loader		
SOF Data	Page_0	<auto>
DE0_Nano_Basic_Computer.sof	EP4CE22F17	

Figure 10. Adding the JTAG Indirect Configuration device.



Device family

- [ ] APEX20K
- [ ] Arria 10
- [ ] Arria GX
- [ ] Arria II GX
- [ ] Arria II GZ
- [ ] Arria V
- [ ] Arria V GZ
- [ ] Cyclone
- [ ] Cyclone II
- [ ] Cyclone III
- [ ] Cyclone III LS
- [x] Cyclone IV E
- [ ] Cyclone IV GX
- [ ] Cyclone V
- [ ] HardCopy II
- [ ] HardCopy III
- [ ] HardCopy IV
- [ ] MAX 10

Device name

- [ ] EP4CE10
- [ ] EP4CE115
- [ ] EP4CE15
- [x] EP4CE22
- [ ] EP4CE30
- [ ] EP4CE40
- [ ] EP4CE55
- [ ] EP4CE6
- [ ] EP4CE75

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.

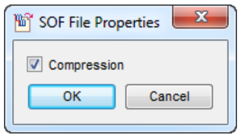


Figure 12. The Properties window.

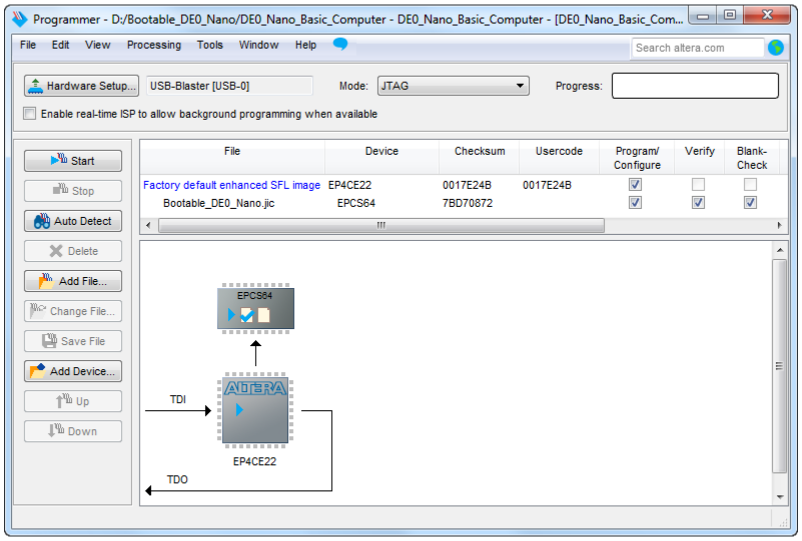
To produce the *Bootable_DE0_Nano.jic* programming file, click the Generate button on the bottom of the screen in Figure 10.

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 II Programmer. Ensure that the Altera Monitor Program software is closed (or disconnected from the DE0-Nano board), because the Quartus II 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 II 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.


Programmer - D:/Bootable_DE0_Nano/DE0_Nano_Basic_Computer - DE0_Nano_Basic_Computer - [DE0_Nano_Basic_Com...
File Edit View Processing Tools Window Help
Search altera.com
Start
Stop
Auto Detect
Delete
Add File...
Change File...
Save File
Add Device...
Up
Down
EPCS64
TDI
TDO
EP4CE22

Hardware Setup... USB-Blaster [USB-0] Mode: JTAG Progress:

☐ Enable real-time ISP to allow background programming when available

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check
Factory default enhanced SFL image	EP4CE22	0017E24B	0017E24B	✓	☐	☐
Bootable_DE0_Nano.jic	EPCS64	7BD70872		✓	✓	✓

Figure 13. The Programmer window.