ERIKA Enterprise Multicore Tutorial

for the Altera Nios II platform

version: 1.0.1
May 27, 2009

EVIDENCE®
EMBEDDING TECHNOLOGY

About Evidence S.r.l.

Evidence is a spin-off company of the ReTiS Lab of the Scuola Superiore S. Anna, Pisa, Italy. We are experts in the domain of embedded and real-time systems with a deep knowledge of the design and specification of embedded SW. We keep providing significant advances in the state of the art of real-time analysis and multiprocessor scheduling. Our methodologies and tools aim at bringing innovative solutions for next-generation embedded systems architectures and designs, such as multiprocessor-on-a-chip, reconfigurable hardware, dynamic scheduling and much more!

Contact Info

Address:
Evidence Srl,
Via Carducci 64/A
Località Ghezzano
56010 S.Giuliano Terme
Pisa - Italy
Tel: +39 050 991 1122, +39 050 991 1224
Fax: +39 050 991 0812, +39 050 991 0855

For more information on Evidence Products, please send an e-mail to the following address: info@evidence.eu.com. Other informations about the Evidence product line can be found at the Evidence web site at: http://www.evidence.eu.com.

Contents

1 RT-Druid and ERIKA Enterprise tutorial for Altera Nios II

This small tutorial describes a set of steps needed to design a simple two CPU multiprocessor application that shows the main features of Erika Enterprise and RT-Druid.

This tutorial suppose that the reader is familiar to design of multiprocessor systems as explained in the Altera document named "Creating Multiprocessor Nios II systems tutorial" available on the Altera web site [1].

The following sections subsumes the Altera tools have been installed under `c:\altera`. This tutorial has been tested on a Stratix 2s60 evaluation board; This tutorial will work with all Nios II evaluation boards, because all the specific multiprocessor hardware involved in the multiprocessor design is a composition of the Altera Avalon Mutex and of the Altera Avalon PIO peripherals, that are provided by Altera on all Nios-II supported FPGAs.

Other tutorials are available for download from the Evidence web site. In particular, we suggest to consider the *API Tutorial*, containing a set of demos working on the `standard` and `full_featured` single-CPU Altera examples showing the behavior of the various Erika Enterprise primitives.

1.1 The hardware design.

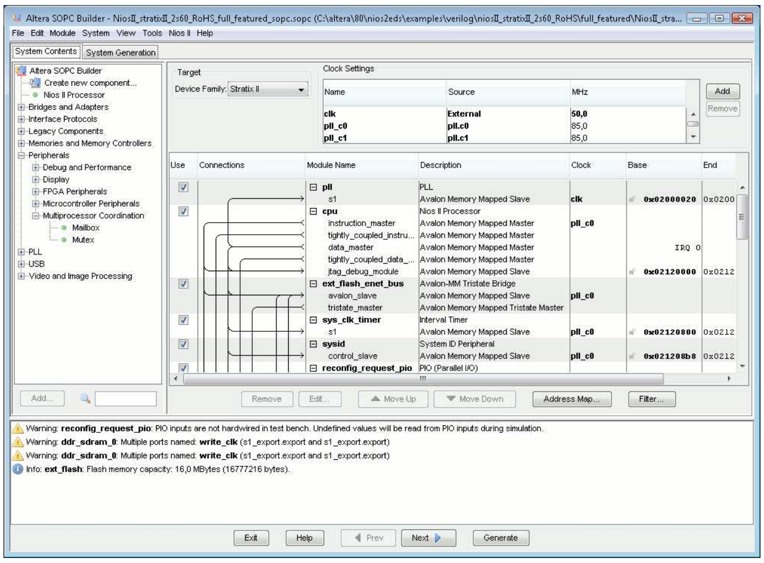
This Section describes how to create a multicore hardware suitable to be used with Erika Enterprise.

The result will be a dual processor design that will include all the peripheral needed for a proper multiprocessor communication; the design will only include standard Altera SOPCBuilder blocks.

The description is done based on the Altera Stratix II 2s60 Rohs Evaluation board. Similar results can be obtained using other Altera evaluation boards.

1. As the first step, copy the entire directory containing the `standard` example from Altera in a separate directory called `evidence_2cpu`. The typical location of the standard example is `c:\altera\80\nios2eds\examples\verilog\boardname\standard` where `boardname` is for example `niosII_cyclone_1c20`. Please create the directory `evidence_2cpu` at the same level of the `standard` example directory.

2. Open Quartus II by double clicking on the `standard.qpf` file inside the `evidence_2cpu` directory just created.


Altera SOPC Builder - NiosII_stratixII_2s60_RoHS_full_featured_sopc.sopc (C:\altera\80\nios2eds\examples\verilog\niosII_stratixII_2s60_RoHS\full_featured\NiosII_stra...
File Edit Module System View Tools Nios II Help
System Contents
System Generation
Altera SOPC Builder
Create new component...
Nios II Processor
Bridges and Adapters
Interface Protocols
Legacy Components
Memories and Memory Controllers
Peripherals
Debug and Performance
Display
FPGA Peripherals
Microcontroller Peripherals
Multiprocessor Coordination
Mailbox
Mutex
PLL
USB
Video and Image Processing
Target
Device Family: Stratix II
Clock Settings
Add
Remove
Add...
Remove
Edit...
Move Up
Move Down
Address Map...
Filter...
Exit
Help
Prev
Next
Generate

Name	Source	MHz
clk	**External**	**50,0**
pll_c0	**pll.c0**	85,0
pll_c1	**pll.c1**	85,0

Use	Connections	Module Name	Description	Clock	Base	End
✓		**pll**	PLL			
		s1	Avalon Memory Mapped Slave	**clk**	**0x02000020**	0x0200
✓		**cpu**	Nios II Processor			
		instruction_master	Avalon Memory Mapped Master	**pll_c0**		
		tightly_coupled_instru...	Avalon Memory Mapped Master			
		data_master	Avalon Memory Mapped Master		IRQ 0	
		tightly_coupled_data_...	Avalon Memory Mapped Master			
		jtag_debug_module	Avalon Memory Mapped Slave		**0x02120000**	0x0212
✓		**ext_flash_enet_bus**	Avalon-MM Tristate Bridge			
		avalon_slave	Avalon Memory Mapped Slave	**pll_c0**		
		tristate_master	Avalon Memory Mapped Tristate Master			
✓		**sys_clk_timer**	Interval Timer			
		s1	Avalon Memory Mapped Slave	**pll_c0**	**0x02120800**	0x0212
✓		**sysid**	System ID Peripheral			
		control_slave	Avalon Memory Mapped Slave	**pll_c0**	**0x021208b8**	0x0212
✓		**reconfig_request_pio**	PIO (Parallel I/O)			

```
Warning: reconfig_request_pio: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: ddr_sdram_0: Multiple ports named: write_clk (s1_export.export and s1_export.export)
Warning: ddr_sdram_0: Multiple ports named: write_clk (s1_export.export and s1_export.export)
Info: ext_flash: Flash memory capacity: 16,0 MBytes (16777216 bytes).
```

Figure 1.1: The SOPCBuilder windows that appears by opening an Altera project.

Note: Some of the new Altera design examples comes with a VHDL/Verilog main file instead of the traditional BDF file. We chose to put the screenshots of the traditional BDF representation of the project, because the graphical representation is easier to understand.

3. Disconnect the pins from the SOPCBuilder Block. Disconnecting the pins will help you in the next steps, because the SOPCBuilder block will change its size after adding the peripherals described in the following paragraphs.

4. Open SOPCBuilder by double clicking on the SOPCBuilder block. A window similar to the one showed in Figure 1.1 appears.

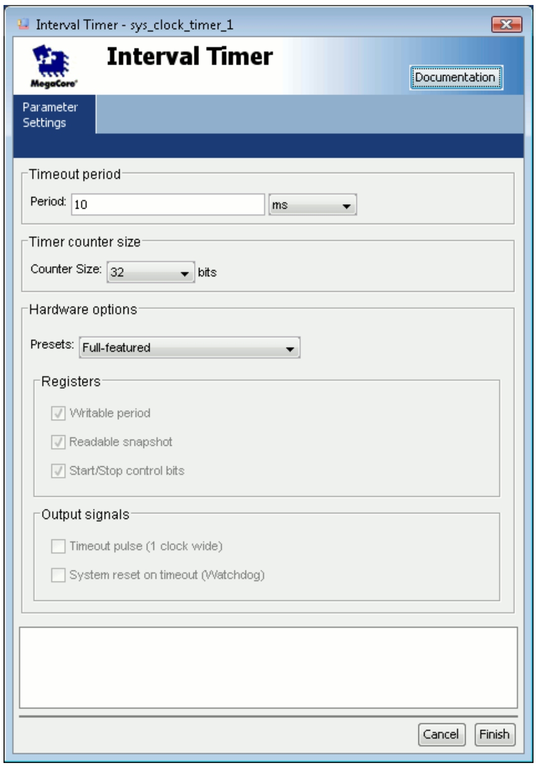
5. Since multiple instances of some SOPCBuilder components will be inserted in the design, change the names of the existing components to represent the CPU to which are connected. To rename a component, right click on the component name and choose *Rename*. First of all, rename the CPU from `cpu` to `cpu_0`.

6. Rename the System Clock Timer from `sys_clk_timer` to `sys_clk_timer_0`.

7. Rename the JTAG UART from `jtag_uart` to `jtag_uart_0`.

8. Rename the High Resolution Timer from `high_res_timer` to `high_res_timer_0`.

9. Add a second CPU named `cpu_1`. In the example, we choose a Nios II/s, with JTAG debug level 2 (you can leave the other options unchanged).


Interval Timer - sys_clock_timer_1

Interval Timer

Timeout period
Period: 10 ms

Timer counter size
Counter Size: 32 bits

Hardware options
Presets: Full-featured

Registers
[✓] Writable period
[✓] Readable snapshot
[✓] Start/Stop control bits

Output signals
[ ] Timeout pulse (1 clock wide)
[ ] System reset on timeout (Watchdog)

Figure 1.2: When creating `sys_clk_timer_1`, use a 10 ms periods.

10. Connect both address and data bus of `cpu_1` to the available memories in the system. Also connect other external buses.

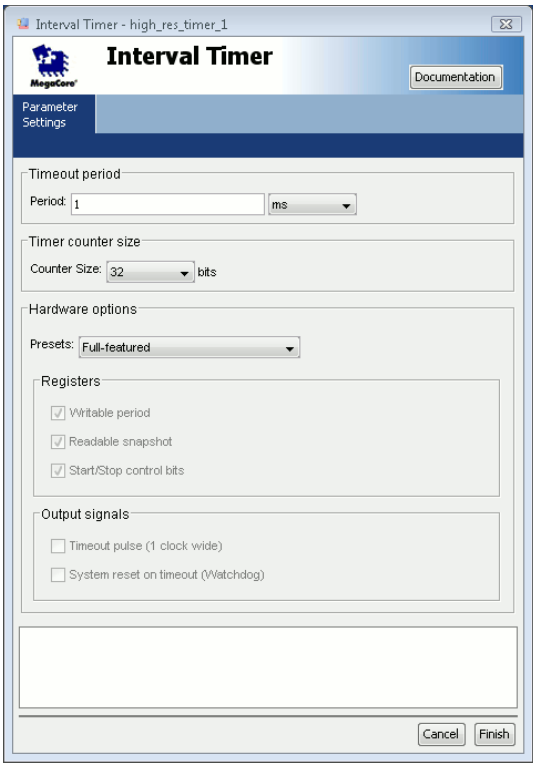
11. Add an Interval Timer component (you can find it under the "Other" tab of SOPCBuilder) named `sys_clk_timer_1`. Set the period to 10 ms. This timer is required by the Altera HAL for the system clock (see Figure 1.2).

12. Add another Interval Timer named `high_res_timer_1`. Leave the proposed options unchanged (see Figure 1.3). This timer is required by the Altera HAL for high resolution time measurement.

13. Add a JTAG Uart (you can find it under the "Communication" tab of SOPCBuilder) named `jtag_uart_1`. Leave the proposed options unchanged. The JTAG UART will be used during the software example to print some messages on the consoles using printf.

14. Connect the two timers and the JTAG UART only to the data master of `cpu_1`.

15. At that point, a basic multiprocessor system has been created. The resulting system is at this point very similar to the one obtained following the Altera Multiprocessor Tutorial [1]. The following three paragraphs describe the actions to setup


Interval Timer - high_res_timer_1

Interval Timer

Timeout period

Period: 1 ms

Timer counter size

Counter Size: 32 bits

Hardware options

Presets: Full-featured

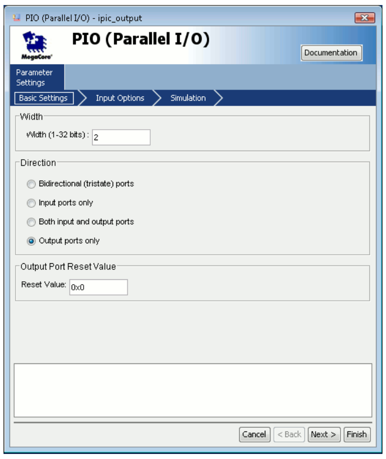
Registers

- [✓] Writable period
- [✓] Readable snapshot
- [✓] Start/Stop control bits

Output signals

- [ ] Timeout pulse (1 clock wide)
- [ ] System reset on timeout (Watchdog)

Figure 1.3: When creating `high_res_timer_1`, use a 1 ms periods.



PIO (Parallel I/O) - ipic_output

PIO (Parallel I/O)

Width

Width (1-32 bits) : 2

Direction

Bidirectional (tristate) ports [✓ Output ports only]

Input ports only

Both input and output ports

Output Port Reset Value

Reset Value: 0x0

Figure 1.4: Output part of the Interprocessor Interrupt. The Figure shows the Avalon PIO output settings.

the SOPCBuilder components needed to implement mutual exclusion between the various CPUs, and to implement the interprocessor interrupt controller.

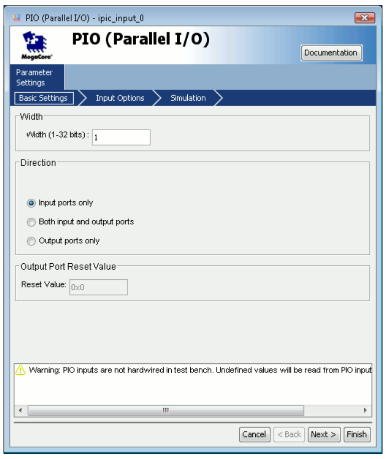
16. Add an Altera Mutex (under the “Other” tab of SOPCBuilder) named `mutex`. Give an “Initial Value” equal to `0x1`, and an “Initial Owner” equal to `0x0`[1]. Connect the Altera Mutex peripheral to the data bus of both CPUs. Please refer to the Erika Enterprise reference manual for Nios II for more information about the usage of the Altera Mutex inside hardware designs compatible with Erika Enterprise.

17. Add an Avalon PIO named `ipic_output`. The output PIO should have an output bit for each CPU in the system. A screenshot of the PIO dialog box can be found in Figure 1.4. This PIO will be used by the two CPUs to send interprocessor interrupts. Connect the PIO to the data master of both CPUs.

18. Add two Avalon PIOs named `ipic_input_0` and `ipic_input_1`. The PIO must be a 1 bit Input PIO, with Synchronous capture on the rising edge, and an Edge IRQ. Screenshots of the PIO dialog boxes can be found in Figures 1.5, 1.6, and 1.7. These PIO will be used by each CPU to receive interprocessor interrupts. Connect `ipic_input_0` to the `cpu_0` data bus, and `ipic_input_1` to the `cpu_1` data bus. This is the last component that have to be added to SOPCBuilder for this demo. Let’s now remove all the errors that appears in the bottom of the window, and setup the other tabs of SOPCBuilder.

19. First of all, IRQ and Base addresses for all the SOPCBuilder peripherals have to be set. To do that, execute the commands “Auto Assign Base Addresses” and “Auto Assign IRQs” under the System menu.

[1]The initial owner has always value `0x0` if you are using Nios II 6.0. Otherwise, it must be the CPUID of the first CPU (in our case, `cpu_0`.



Width

Width (1-32 bits) : 1

Direction

[✓ Input ports only]

Both input and output ports

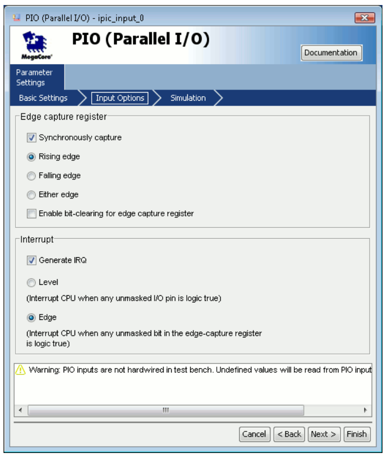
Output ports only

Output Port Reset Value

Reset Value: 0x0

Warning: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO input

Figure 1.5: Input part of the Interprocessor Interrupt. The Figure shows the Avalon PIO basic settings.



Edge capture register

[✓ Synchronously capture]

[✓ Rising edge]

Falling edge

Either edge

Enable bit-clearing for edge capture register

Interrupt

[✓ Generate IRQ]

Level

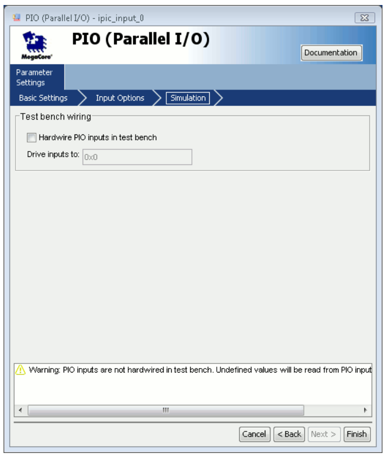
(Interrupt CPU when any unmasked I/O pin is logic true)

[✓ Edge]

(Interrupt CPU when any unmasked bit in the edge-capture register is logic true)

Warning: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO input

Figure 1.6: Input part of the Interprocessor Interrupt. The Figure shows the Avalon PIO input settings.



Test bench wiring

[ ] Hardwire PIO inputs in test bench

Drive inputs to: 0x0

Warning: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO input

Figure 1.7: Input part of the Interprocessor Interrupt. The Figure shows the Avalon PIO simulation settings.

20. If the design you are developing includes a component connected to the external bus, then you have to verify the connections of interrupt lines that may be connected to more than one CPU.

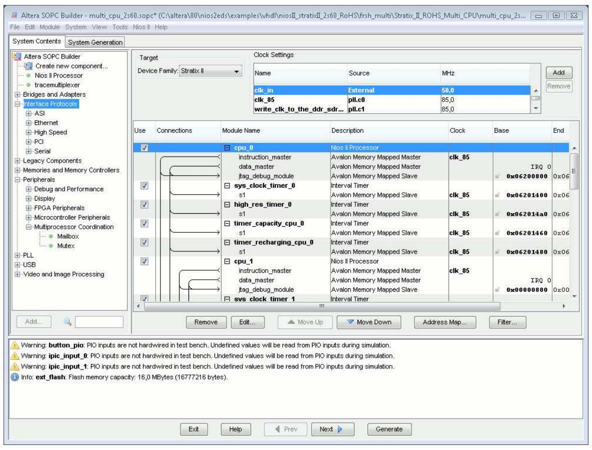
> **Warning:** The interrupt lines of every Altera SOPCBuilder component have to be connected to only one CPU!

    The `standard` example for Stratix 1s40 Evaluation board contains an instance of the LAN91c111 SOPCBuilder component. When adding the second CPU `cpu_1`, SOPCBuilder allows the user to set an interrupt line for both CPUs. To solve the problem, you have to set the interrupt priority of the `cpu_1` connection to "NC" (Not Connected). A hand-made renumbering of the interrupts may also be optionally done.

21. Figure 1.8 shows the various components and the interrupt connections (the picture was taken after removing all the errors in the bottom part of the window as discussed in the following paragraphs). The following paragraphs discuss the settings of the "Board Settings" and the "Nios II CPU Settings" tabs of SOPCBuilder.

22. Looking at the Board Settings tab, the PIO components used for the input and output part of the interprocessor interrupt generate a set of unassigned pins. To remove the error, if you are using Nios II version 5.0, set the assignment of these pins to "No Assignment"; if you are using Nios II 5.1 or 6.0, as shown in Figure 1.9, set the assignment of these pins to "Assign in Quartus II Project".

23. As the last thing, set the Reset and Exception address of the Nios II CPUs as in Figure 1.10 and 1.11. In this example, the values are set considering the following rules:



Target

Device Family: Stratix II

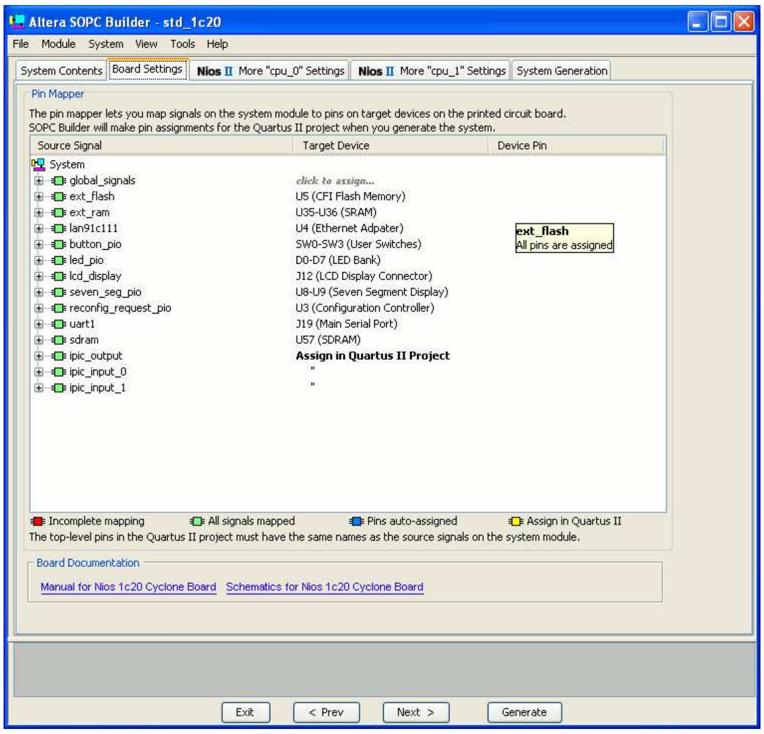
Clock Settings

Name	Source	MHz
clk_in	External	50,0
clk_85	pll.c0	85,0
write_clk_to_the_ddr_sdr...	pll.c1	85,0

Use	Connections	Module Name	Description	Clock	Base	End
✓		cpu_0	Nios II Processor			
		instruction_master	Avalon Memory Mapped Master	clk_85		
		data_master	Avalon Memory Mapped Master		IRQ 0	
		jtag_debug_module	Avalon Memory Mapped Slave		0x06200800	0x06
✓		sys_clock_timer_0	Interval Timer			
		s1	Avalon Memory Mapped Slave	clk_85	0x06201400	0x06
✓		high_res_timer_0	Interval Timer			
		s1	Avalon Memory Mapped Slave	clk_85	0x062014a0	0x06
✓		timer_capacity_cpu_0	Interval Timer			
		s1	Avalon Memory Mapped Slave	clk_85	0x06201460	0x06
✓		timer_recharging_cpu_0	Interval Timer			
		s1	Avalon Memory Mapped Slave	clk_85	0x06201480	0x06
✓		cpu_1	Nios II Processor			
		instruction_master	Avalon Memory Mapped Master	clk_85		
		data_master	Avalon Memory Mapped Master		IRQ 0	
		jtag_debug_module	Avalon Memory Mapped Slave		0x00000800	0x00
✓		sys_clock_timer_1	Interval Timer			

Warning: **button_pio**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: **ipic_input_0**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Warning: **ipic_input_1**: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Info: **ext_flash**: Flash memory capacity: 16,0 MBytes (16777216 bytes).

Figure 1.8: The list of SOPCBuilder components composing this demo.


Altera SOPC Builder - std_1c20
File Module System View Tools Help
System Contents
Board Settings
Nios II More "cpu_0" Settings
Nios II More "cpu_1" Settings
System Generation
Exit
< Prev
Next >
Generate

Pin Mapper

The pin mapper lets you map signals on the system module to pins on target devices on the printed circuit board.
SOPC Builder will make pin assignments for the Quartus II project when you generate the system.

Source Signal	Target Device	Device Pin
System		
global_signals	click to assign...	
ext_flash	U5 (CFI Flash Memory)	
ext_ram	U35-U36 (SRAM)	
lan91c111	U4 (Ethernet Adpater)	
button_pio	SW0-SW3 (User Switches)	
led_pio	D0-D7 (LED Bank)	
lcd_display	J12 (LCD Display Connector)	
seven_seg_pio	U8-U9 (Seven Segment Display)	
reconfig_request_pio	U3 (Configuration Controller)	
uart1	J19 (Main Serial Port)	
sdram	U57 (SDRAM)	
ipic_output	**Assign in Quartus II Project**	
ipic_input_0	"	
ipic_input_1	"	

ext_flash
All pins are assigned

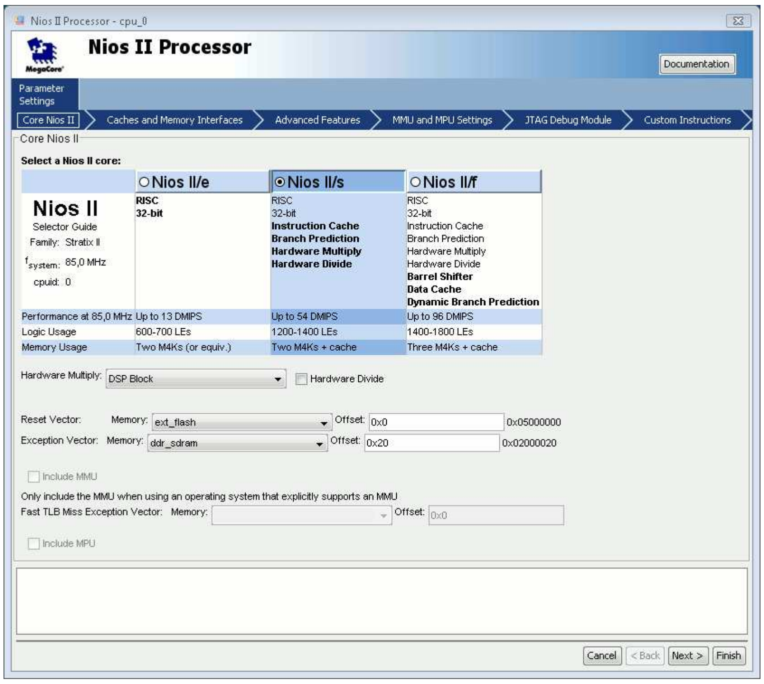
🟥 Incomplete mapping 🟩 All signals mapped 🟦 Pins auto-assigned 🟨 Assign in Quartus II

The top-level pins in the Quartus II project must have the same names as the source signals on the system module.

Board Documentation

Manual for Nios 1c20 Cyclone Board Schematics for Nios 1c20 Cyclone Board

Figure 1.9: The Board Settings tab after assigning the Interprocessor interrupt pins (Nios II Version 5.1 and 6.0).



Core Nios II

Select a Nios II core:

Nios II Selector Guide Family: Stratix II f_{system}: 85,0 MHz cpuid: 0	○ Nios II/e	● Nios II/s	○ Nios II/f
	RISC **32-bit**	RISC 32-bit **Instruction Cache** **Branch Prediction** **Hardware Multiply** **Hardware Divide**	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide **Barrel Shifter** **Data Cache** **Dynamic Branch Prediction**
Performance at 85,0 MHz	Up to 13 DMIPS	Up to 54 DMIPS	Up to 96 DMIPS
Logic Usage	600-700 LEs	1200-1400 LEs	1400-1800 LEs
Memory Usage	Two M4Ks (or equiv.)	Two M4Ks + cache	Three M4Ks + cache

Hardware Multiply: DSP Block ☐ Hardware Divide

Reset Vector: Memory: ext_flash Offset: 0x0 0x05000000
Exception Vector: Memory: ddr_sdram Offset: 0x20 0x02000020

☐ Include MMU

Only include the MMU when using an operating system that explicitly supports an MMU
Fast TLB Miss Exception Vector: Memory: Offset: 0x0

☐ Include MPU

Figure 1.10: The settings for the `cpu_0`.

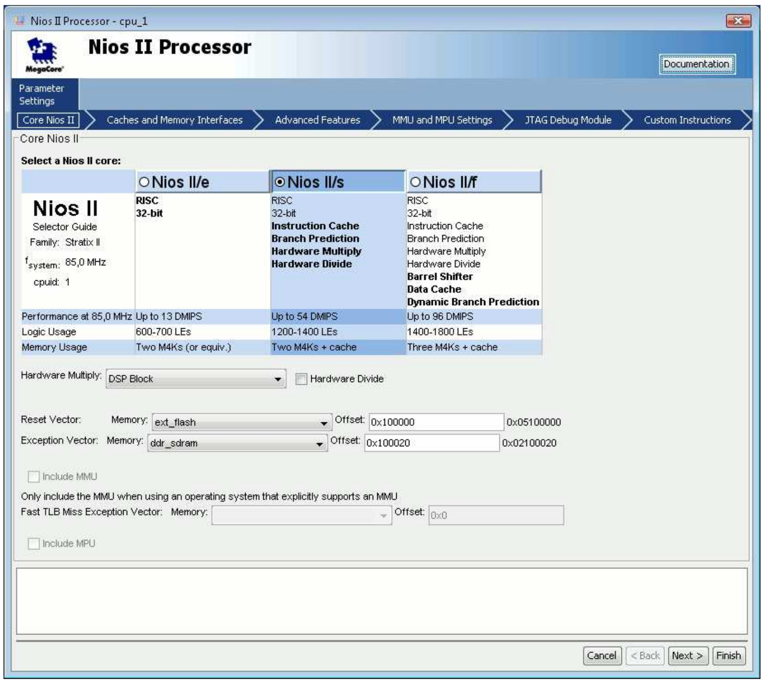
- The Reset addresses of both CPUs are set to the external flash;
- The Exception address of both CPUs are set to SDRAM;
- The Reset and Exception addresses of `cpu_0` are set to the start of the respective memories;
- The Reset and Exception addresses of `cpu_1` are set around the middle of the address space of the respective Flash and SDRAM memories.
- The Reset is always placed at the startup of a flash block (see Section 1.8).

24. Generate the SOPCBuilder Block.

25. As the result of the generation, a .PTF file is created. If you are using nios II version 5.0 or 5.1, please check inside the PTF File that `cpu_0` has CPUID 0, and that `cpu_1` has CPUID 1. That is important, because as explained in the Erika Enterprise Reference Manual, influences the Startup Barrier behavior. The check is not needed if you are using nios II 6.0.

26. Go back to Quartus II, and update the symbol. Connect back all the various pins of the `standard` design, to their respective components.

27. Connect the PIO components created for the interprocessor interrupt as shown in Figure 1.12. Each pin of the output PIO have to be connected to the correspondent CPU. The usage of the named pins of Quartus II simplifies the connection.


Nios II Processor - cpu_1

Nios II Processor

Parameter Settings

Core Nios II > Caches and Memory Interfaces > Advanced Features > MMU and MPU Settings > JTAG Debug Module > Custom Instructions

Core Nios II

Select a Nios II core:

Nios II Selector Guide Family: Stratix II f_{system}: 85,0 MHz cpuid: 1	Nios II/e	[✓ Nios II/s]	Nios II/f
	RISC 32-bit	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide Barrel Shifter Data Cache Dynamic Branch Prediction
Performance at 85,0 MHz	Up to 13 DMIPS	Up to 54 DMIPS	Up to 96 DMIPS
Logic Usage	600-700 LEs	1200-1400 LEs	1400-1800 LEs
Memory Usage	Two M4Ks (or equiv.)	Two M4Ks + cache	Three M4Ks + cache

Hardware Multiply: DSP Block Hardware Divide

Reset Vector: Memory: ext_flash Offset: 0x100000 0x05100000

Exception Vector: Memory: ddr_sdram Offset: 0x100020 0x02100020

Include MMU

Only include the MMU when using an operating system that explicitly supports an MMU

Fast TLB Miss Exception Vector: Memory: Offset: 0x0

Include MPU

Figure 1.11: The settings for the `cpu_1`.

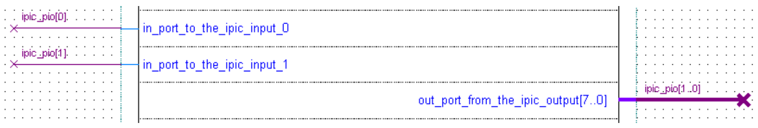

ipic_pio[0]
ipic_pio[1]
in_port_to_the_ipic_input_0
in_port_to_the_ipic_input_1
out_port_from_the_ipic_output[7..0]
ipic_pio[1..0]

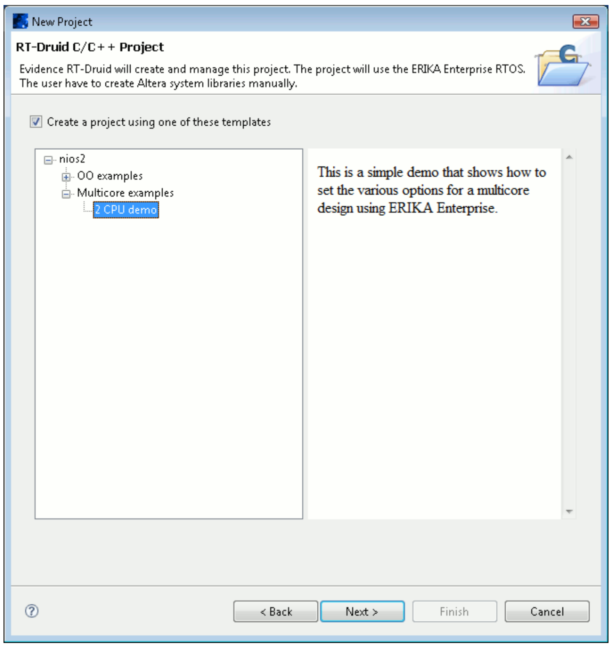
Figure 1.12: Connecting the output part of the Interprocessor interrupt to the input part of the Interprocessor interrupt.

Basically, pin 0 of the output PIO have to be connected to the CPU with `CPUID` 0 (typically, `cpu_0`); pin 1 of the output PIO have to be connected to the CPU with `CPUID` 1 (typically, `cpu_1`), and so on.

28. Finally, you can compile the Quartus II design to produce your first design compatible with Erika Enterprise.

1.2 Creating the Altera System Libraries.

Erika Enterprise applications uses the Altera System Libraries as the base for linker scripts, boot code and device drivers. This Section shortly describes which are the main steps to create the System Libraries needed to be linked to the tutorial application. For more informations on Altera System Libraries, please refer to official Altera Nios II documentation.


New Project

RT-Druid C/C++ Project

Evidence RT-Druid will create and manage this project. The project will use the ERIKA Enterprise RTOS. The user have to create Altera system libraries manually.

☑ Create a project using one of these templates

- nios2
  - OO examples
  - Multicore examples
    - 2 CPU demo

This is a simple demo that shows how to set the various options for a multicore design using ERIKA Enterprise.

Figure 1.13: Selecting the Multicore template.

Open the Nios II IDE from the last SOPCBuilder tab, and select “New” from the File menu, and then “Project...”. Choose “System Library” from the Altera Nios II tab of the New Project Dialog box. Name the system library `evidence_2cpu_cpu0`; the PTF file of the hardware project should be already selected. Be sure that `cpu_0` is selected.

Repeat the above steps for the second CPU `cpu_1`, naming the second System Library Project as `evidence_2cpu_cpu1`, and selecting `cpu_1`.

Build the two system libraries right-clicking on the project name and choosing “Build Project”.

1.3 The RT-Druid Project

Open the Nios II IDE from SOPCBuilder, and select “New Project...” from the File menu. Choose “RT-Druid Nios Project” from the Evidence tab of the New Project Dialog box. A dialog box appears. Choose the 2 CPu demo, as shown in Figure ??. Name the project `evidence_2cpu` and press the Finish button.

1.4 Updating the OIL File

This Section asks you to change the OIL file inserting the locations of the system libraries you just created. You need to do that only if the system library names and locations you chose are not the ones specified in this tutorial. You can go directly to the next Section if you chose the file names specified early in this tutorial.

Inside the OIL file, look for the `CPU_DATA` sections located at the top of the OIL description inside the `OS` section. There is a `CPU_DATA` section for each CPU in the system. Each section has two settings, called `SYSTEM_LIBRARY_NAME` and `SYSTEM_LIBRARY_PATH`. You need to change them with the real names of the system libraries. When specifying the pathnames, please use the slash (`/`) character and not the backslash (`\`) character. These two settings will be used in the makefiles that are automatically generated by RT-Druid[2].

1.5 Compiling the application

Right click on the project name, and select "Build Project". The demo application will be compiled, and two ELF binaries will be produced.

1.6 Running the application

First of all, check if the "Allow multiple active run/debug sessions" option has been enabled in the Nios II preferences (see Figure 1.14).

Then, click on the "Run..." option in the toolbar. A dialog box appears allowing the specification of the Debug configurations. Create a Nios II Hardware configuration for each CPU in the system. For each configuration:

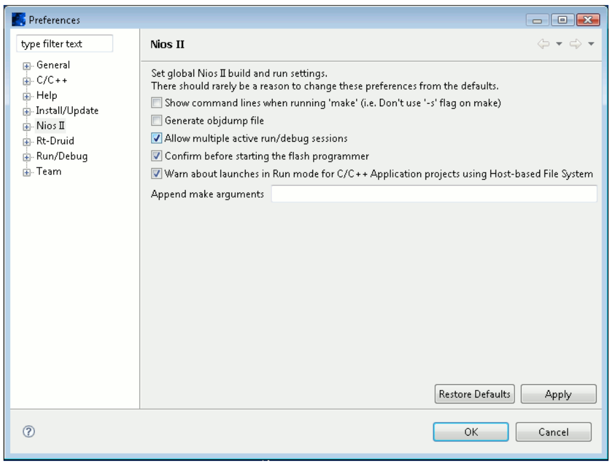
- in the Target Hardware frame, select the project PTF file, and the right CPU in the system;
- in the Nios II ELF Executable text box, please select the Nios II ELF executable for the selected CPU;
- do not forget to click on the "Target Connection" tab and check that the correct JTAG UART has been automatically selected.

Finally, create a Nios II Multiprocessor collection grouping all the Hardware configurations just created.

You are now ready to start the multiprocessor application. Every time you want to run a Multiprocessor application, you have to:

[2]If unsure on the value to put in these variables, you can do in one of the following two ways:

- create a normal Altera project for one of the CPUs (for example, a Hello World application), linking it to the System library you want to use on the particular CPU. Then, compile the Altera application, and look at the file `Debug/makefile` inside the Altera project. The value of the `SYSTEM_NAME` variable is the value that have to be put in the OIL tag `SYSTEM_LIBRARY_NAME`; the value of the variable `SYSTEM_DIR` is the value of the OIL tag `SYSTEM_LIBRARY_DIR`.
- a faster way to fill the `SYSTEM_LIBRARY_PATH` variable is to open the properties of the Altera System Libraries, and then look at the "Location" value inside the info tab. Note that you have to substitute `\` with `/`.



Nios II

Set global Nios II build and run settings.
There should rarely be a reason to change these preferences from the defaults.

- [ ] Show command lines when running 'make' (i.e. Don't use '-s' flag on make)
- [ ] Generate objdump file
- [✓] Allow multiple active run/debug sessions
- [✓] Confirm before starting the flash programmer
- [✓] Warn about launches in Run mode for C/C++ Application projects using Host-based File System

Append make arguments

Figure 1.14: Enabling multiprocessor debugging in the Nios II preferences.

- open the Quartus II Programmer under the Tools menu, and program the SOF file you find in the Hardware project directory.

Warning: You need to reprogram the FPGA every time you start a debug session, because the Altera Mutex peripheral is not reset to its initial value upon a Debug Stop.

- choose the Multiprocessor Collection Debug configuration when pressing the "Run..." button.

When running the application, the following behavior happens:

- Both CPUs start in Debug mode (or Run mode if you selected that).
- Both CPUs print a message like the following one:

```
Hello from CPU 0!

Press a button to activate the tasks...
```

- The two CPUs synchronize at the startup barrier inside the `StartOS()` primitive inside `main()`.
- When both CPUs enters the `StartOS()` primitive, the synchronization barrier is passed, and the `StartOS()` primitive returns.
- Press one of the buttons on the evaluation board. The press of the button provoke the activation of three threads. Each thread prints a message on the console of the CPU where it is allocated to. In the configuration shipped with Erika Enterprise task 0 and 1 are allocated on CPU 0, whereas task 2 is allocated to CPU 1.

1.7 Partitioning the software

RT-Druid and Erika Enterprise allow an easy partitioning of the application software. Changing the partitioning of the tasks into the CPUs is very simple: you just need to change the `CPU_ID` parameter in the task section of the OIL file.

For example, you can put all the three tasks to CPU 0, or the tasks to CPU 1, or choose an intermediate partitioning. In all these cases, multiprocessor resource handling and task activations will be hided by Erika Enterprise, without changing the application software. Please note that a different partitioning scheme does not require a change in the application source code, but only in the OIL configuration file.

To test a different partitioning, just change the OIL File, recompile and rerun the demo.

1.8 Flashing the demo on the evaluation board

The last phase in the development of a multiprocessor design using Nios II is typically the flashing of the demo on the flash device that is usually present in the development (or production) board.

To do that, open the Flash Programmer (under the Tools menu of the Nios II IDE), and create one configuration for each CPU. Each configuration for each CPU should specify the PTF file of the multiprocessor design, selecting the appropriate cpu and ELF file. Moreover, the first CPU should also include the flashing of the SOF file. Figure 1.15 and 1.15 show a typical configuration of the flash programmer for the CPU 0 and 1 of the 2 CPU demo showed in this tutorial. As you can see, the SOF file is only included in the flash configuration of CPU 0.

Please note that this way of flashing the data to the evaluation board flash memory depends on the reset addresses used for the CPUs in SOPCBuilder. The reset addresses, in particular, have been chosen to have non-overlapping flash regions for each CPU.

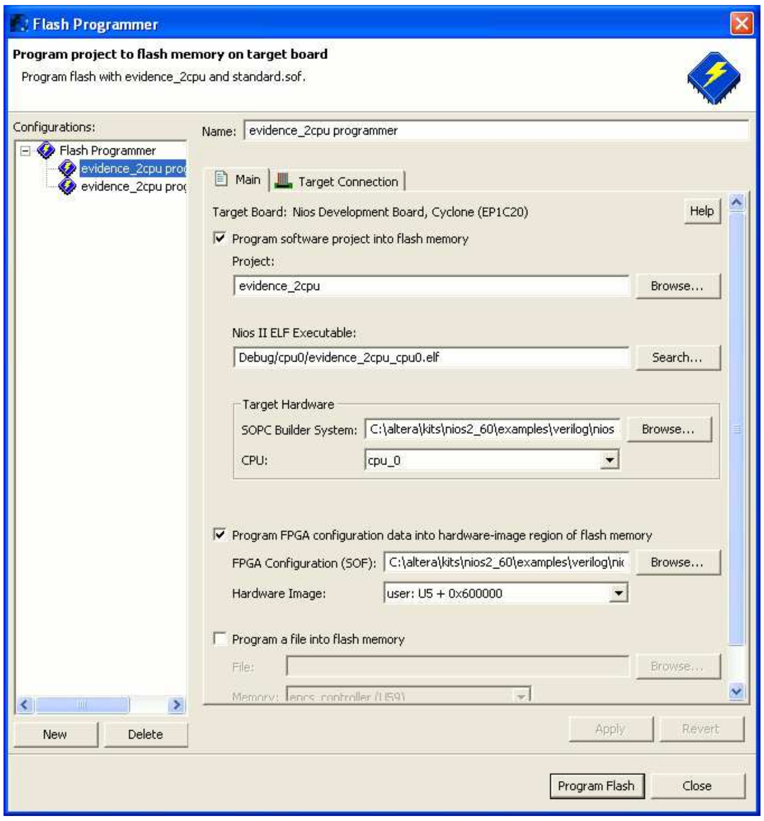
Warning: Some care have to be used when defining the addresses that will be used in the flash.

Flash memories, in fact, are divided in blocks. The flash memory programming model says that flash devices can be written and erased with a granularity of a block.

When writing the software for the Nios II platform, each CPU produces a separate ELF file, that is typically programmed separately using the Altera Flash Programmer tool.

It is important that each flash block contains data coming from a single ELF image. To do that, reset addresses of each CPU in SOPCBuilder have to be set preferably to the starting address of a flash block. Figure 1.17 gives an example of the erroneous situation.

As an alternative, the various flash images from each CPU have to be packed together in a single flash programming file.



Program project to flash memory on target board
Program flash with evidence_2cpu and standard.sof.

Name: evidence_2cpu programmer

Target Board: Nios Development Board, Cyclone (EP1C20)

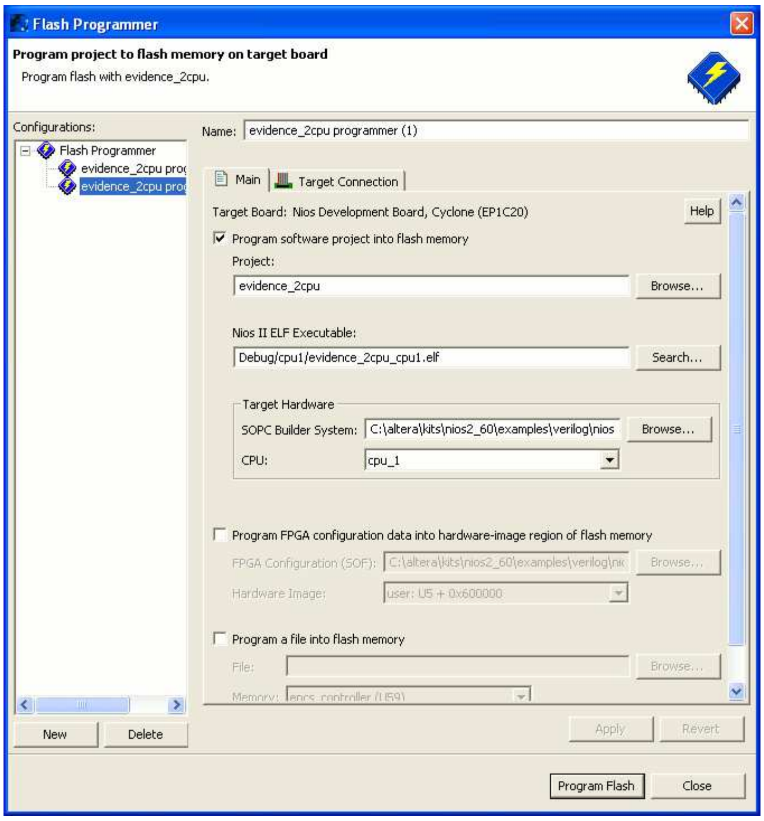
☑ Program software project into flash memory
Project: evidence_2cpu
Nios II ELF Executable: Debug/cpu0/evidence_2cpu_cpu0.elf

Target Hardware
SOPC Builder System: C:\altera\kits\nios2_60\examples\verilog\nios
CPU: cpu_0

☑ Program FPGA configuration data into hardware-image region of flash memory
FPGA Configuration (SOF): C:\altera\kits\nios2_60\examples\verilog\nio
Hardware Image: user: U5 + 0x600000

☐ Program a file into flash memory
File:
Memory: epcs_controller (U59)

Figure 1.15: The Flash Programmer configuration for CPU 0.



Flash Programmer

Program project to flash memory on target board

Program flash with evidence_2cpu.

Configurations:
- Flash Programmer
  - evidence_2cpu prog
  - evidence_2cpu prog

Name: evidence_2cpu programmer (1)

Main | Target Connection

Target Board: Nios Development Board, Cyclone (EP1C20)

☑ Program software project into flash memory

Project: evidence_2cpu

Nios II ELF Executable: Debug/cpu1/evidence_2cpu_cpu1.elf

Target Hardware

SOPC Builder System: C:\altera\kits\nios2_60\examples\verilog\nios

CPU: cpu_1

☐ Program FPGA configuration data into hardware-image region of flash memory

FPGA Configuration (SOF): C:\altera\kits\nios2_60\examples\verilog\nic

Hardware Image: user: U5 + 0x600000

☐ Program a file into flash memory

File:

Memory: epcs_controller (U59)

Figure 1.16: The Flash Programmer configuration for CPU 1.

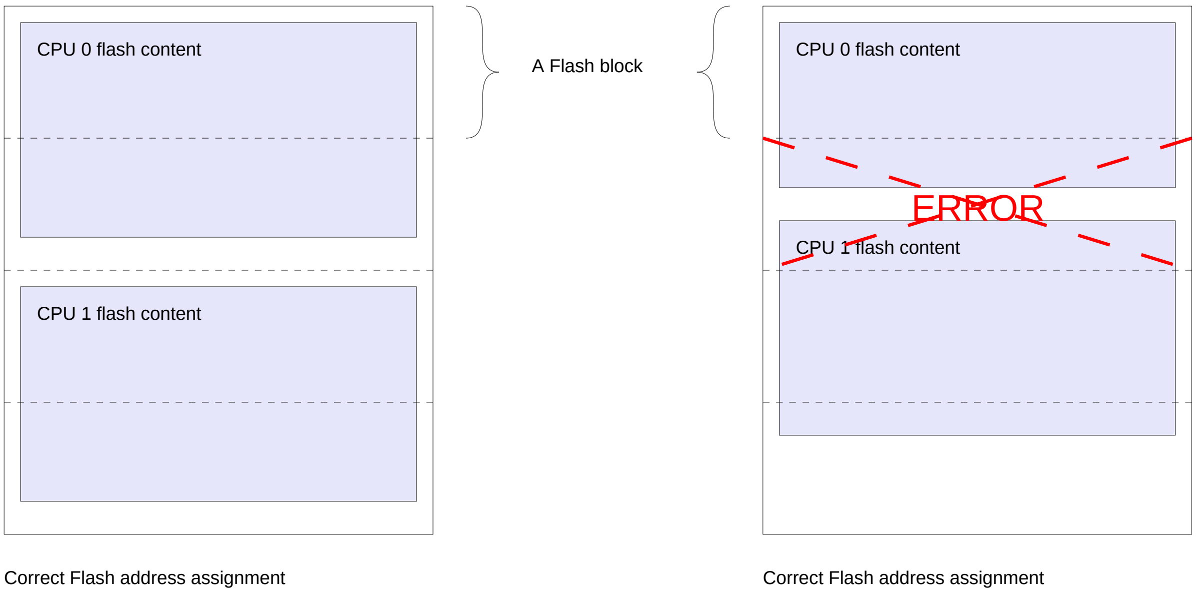


Figure 1.17: This Figure shows a typical flash memory layout highlighting the error situation that appears when a flash block contains data from two CPUs.

> **Warning:** If you are using onchip memories, after compiling the software to be flashed, remember to run again the Quartus II Assembler to include the generated hex files for these memories inside the SOF file.

After that, run the flashing procedure from the Altera Flash Programmer. The following lines show a typical output of the flash programmer.

For CPU 0:

```
#! /bin/sh
#
# This file was automatically generated by the Nios II IDE Flash Programmer.
#
# It will be overwritten when the flash programmer options change.
#

cd c:/altera/kits/nios2/bin/eclipse/workspace3/demo_2cpu/Debug/cpu0

# Creating .flash file for the FPGA configuration
$SOPC_KIT_NIOS2/bin/sof2flash --flash=U5 --offset=0x400000
  --input=$SOPC_KIT_NIOS2/examples/verilog/niosII_stratix_1s40/evidence_2cpu/standard.sof
  --output=standard.flash
Info: *******************************************************************
Info: Running Quartus II Convert_programming_file
Info: Command: quartus_cpf --no_banner --convert
  C:/altera/kits/nios2/examples/verilog/niosII_stratix_1s40/evidence_2cpu/standard.sof
  standard.rbf
Info: Quartus II Convert_programming_file was successful. 0 errors, 0 warnings
    Info: Processing ended: Thu Oct 27 09:36:08 2005
    Info: Elapsed time: 00:00:02

# Programming flash with the FPGA configuration
$SOPC_KIT_NIOS2/bin/nios2-flash-programmer --input=standard.flash
  --sof=$SOPC_KIT_NIOS2/components/altera_nios_dev_board_stratix_1s40/system/
    altera_nios_dev_board_stratix_1s40.sof
  --base=0x00800000
27-ott-2005 9.36.13 - (INFO) nios2-flash-programmer:
  Launching Quartus Programmer to download:
     C:/altera/kits/nios2/components/altera_nios_dev_board_stratix_1s40/system/
       altera_nios_dev_board_stratix_1s40.sof
Pre-Reading 1520KBytes of data from U5:
    |----.----+----.----|
    ********************* (12.312 sec).
Erasing 24 Sectors:
    |----.----+----.----|
    ********************* (15.61 sec).
Writing 1536KBytes :
    |----.----+----.----|
    ********************* (48.0 sec).
Verifying 1536KBytes of data:
    |----.----+----.----|
    ********************* (11.593 sec).
27-ott-2005 9.37.49 - (INFO) nios2-flash-programmer:
  Success. Verified 1536Kbytes written to U5.
27-ott-2005 9.37.49 - (INFO) nios2-flash-programmer:
  Flash programming complete

# Creating .flash file for the project
$SOPC_KIT_NIOS2/bin/elf2flash --flash=U5 --base=0x00000000 --end=0x7fffff
  --reset=0x0 --input=demo_2cpu_cpu0.elf --output=ext_flash.flash
  --boot=$SOPC_KIT_NIOS2/components/altera_nios2/boot_loader_cfi.srec

# Programming flash with the project
```

```
$SOPC_KIT_NIOS2/bin/nios2-flash-programmer --input=ext_flash.flash
  --sof=__NO_SOF_PLEASE__ --base=0x00800000
27-ott-2005 9.37.50 - (INFO) nios2-flash-programmer: SOF-download skipped.
Pre-Reading 72KBytes of data from U5:
    |----.----+----.----|
    ********************* (1.125 sec).
Erasing 2 Sectors:
    |----.----+----.----|
    ********************* (1.438 sec).
Writing 128KBytes :
    |----.----+----.----|
    ********************* (3.609 sec).
Verifying 128KBytes of data:
    |----.----+----.----|
    ********************* (1.312 sec).
27-ott-2005 9.38.00 - (INFO) nios2-flash-programmer:
  Success. Verified 128Kbytes written to U5.
27-ott-2005 9.38.00 - (INFO) nios2-flash-programmer:
  Flash programming complete
```

For CPU 1:

```
#! /bin/sh
#
# This file was automatically generated by the Nios II IDE Flash Programmer.
#
# It will be overwritten when the flash programmer options change.
#

cd c:/altera/kits/nios2/bin/eclipse/workspace3/demo_2cpu/Debug/cpu1

# Creating .flash file for the project
$SOPC_KIT_NIOS2/bin/elf2flash --flash=U5 --base=0x00000000 --end=0x7fffff --reset=0x100000
  --input=demo_2cpu_cpu1.elf --output=ext_flash.flash
  --boot=$SOPC_KIT_NIOS2/components/altera_nios2/boot_loader_cfi.srec

# Programming flash with the project
$SOPC_KIT_NIOS2/bin/nios2-flash-programmer --input=ext_flash.flash
  --sof=$SOPC_KIT_NIOS2/components/altera_nios_dev_board_stratix_1s40/system/
    altera_nios_dev_board_stratix_1s40.sof --base=0x00800000
27-ott-2005 9.39.54 - (INFO) nios2-flash-programmer:
  Launching Quartus Programmer to download:
     C:/altera/kits/nios2/components/altera_nios_dev_board_stratix_1s40/system/
       altera_nios_dev_board_stratix_1s40.sof
Pre-Reading 66KBytes of data from U5:
    |----.----+----.----|
    ********************* (1.125 sec).
Erasing 2 Sectors:
    |----.----+----.----|
    ********************* (1.86 sec).
Writing 128KBytes :
    |----.----+----.----|
    ********************* (3.531 sec).
Verifying 128KBytes of data:
    |----.----+----.----|
    ********************* (1.203 sec).
27-ott-2005 9.40.09 - (INFO) nios2-flash-programmer:
  Success. Verified 128Kbytes written to U5.
27-ott-2005 9.40.09 - (INFO) nios2-flash-programmer:
  Flash programming complete
```

1.9 Running the demo from flash without the Nios II IDE

Once the SOF file with the FPGA setup and the ELF file with the software of each CPU has been stored into the flash memory of the evaluation board, you can run the demo without the need of the Altera Nios II IDE.

To do that, follow these steps:

1. Connect the USB-Blaster to the board.
2. Turn on the board. The demo starts (on a Stratix 1s40 evaluation board, with the Design file created in this tutorial, the result is that the two 8-segments digits are all on).
3. Open two *Nios II SDK shell*.
4. On the first shell, execute the command `nios2-terminal --instance=1`. As a result, the terminal connects to the JTAG UART on CPU 0, displaying the hello message of CPU 0.

Warning: The JTAG chains assigned to each JTAG UART instance may vary with the designs.

5. On the second shell, execute the command `nios2-terminal --instance=0`. As a result, the terminal connects to the JTAG UART on CPU 1, displaying the hello message of CPU 1.
6. Press the buttons on the board to get the same output you got when debugging the demo on the Nios II IDE.
7. By pressing the *Reset CPU* button, the demo starts again, displaying again the hello message on both terminals.

2 History

Version	Comment
Initial version	It was Chapter 2 of the Erika Enterprise Reference Manual version 1.2.4.
1.0.0	Updated text and screenshots to Nios II 6.0.
1.0.1	Added new versioning mechanism.
1.1.0	Support for Nios II 8.0.

Bibliography

[1] Altera Corporation. Creating multiprocessor nios ii systems tutorial. Nios II literature page, http://www.altera.com/literature/lit-nio2.jsp, 2005.