Using Nios II Tightly Coupled Memory

Tutorial
## Contents

**About This Tutorial** ................................................................. V  
How to Find Information .......................................................... v  
How to Contact Altera .............................................................. vi  
Typographical Conventions ...................................................... vii  

**Using Tightly Coupled Memory** ............................................. 9  
Introduction ................................................................................ 9  
  Reasons for Using Tightly Coupled Memory ............................ 9  
  Trade Offs ............................................................................. 10  
Guidelines for Using Tightly Coupled Memory .......................... 10  
  Hardware Guidelines ............................................................ 10  
  Software Guidelines ............................................................ 11  
  Locating Functions in Tightly Coupled Memory ....................... 11  
Tightly Coupled Memory Interface .......................................... 12  
  Restrictions .......................................................................... 12  
  Dual Port Memories ......................................................... 13  
Building a Nios II System with Tightly Coupled Memory ........ 13  
  Hardware and Software Requirements ................................. 14  
  Modify the Example Design to Include Tightly Coupled Memories ... 15  
  Create the Tightly Coupled Memories ................................... 17  
  Connect & Position the Tightly Coupled Memories ............... 21  
  Exception Address Selection .............................................. 23  
  Add a Performance Counter .............................................. 24  
  Build the Hardware System .............................................. 25  
  Create a Software Project to Exercise Tightly Coupled Memories .... 26  
  Specify a Separate Exception Stack .................................. 26  
  Change the Projects’ Build Properties ................................. 28  
Executing the Software on the Modified Standard_tcm Hardware Design ........................................... 28  
  Download and Run Tcm_vs_cache Project ............................ 29  
  Interrupt Response Time ................................................... 29  
  Memory Access Times ...................................................... 30  
Software Example Implementation Notes .................................. 30  
  Checksum Memory Block Size ...................................... 30  
  Heating the Instruction Cache ....................................... 31  
  Unrolled Checksum Loops for Optimization ......................... 31  
Conclusion ............................................................................. 31
This tutorial describes how to use tightly coupled memory in Nios® II designs.

The following table shows the revision history of this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>July 2005</td>
<td>First publication.</td>
</tr>
</tbody>
</table>

**How to Find Information**

- The Adobe Acrobat Find feature allows you to search the contents of a PDF file. Click the binoculars toolbar icon to open the Find dialog box.
- Bookmarks serve as an additional table of contents.
- Thumbnail icons, which provide miniature previews of each page, provide a link to the pages.
- Numerous links, shown in green text, allow you to jump to related information.
How to Contact Altera

For the most up-to-date information about Altera® products, go to the Altera world-wide web site at www.altera.com. For technical support on this product, go to www.altera.com/mysupport. For additional information about Altera products, consult the sources shown below.

<table>
<thead>
<tr>
<th>Information Type</th>
<th>USA &amp; Canada</th>
<th>All Other Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical support</td>
<td><a href="http://www.altera.com/mysupport/">www.altera.com/mysupport/</a></td>
<td>altera.com/mysupport/</td>
</tr>
<tr>
<td></td>
<td>(800) 800-EPLD (3753)</td>
<td>+1 408-544-7000</td>
</tr>
<tr>
<td></td>
<td>(7:00 a.m. to 5:00 p.m.</td>
<td>(7:00 a.m. to 5:00 p.m. (GMT-8:00) Pacific Time</td>
</tr>
<tr>
<td></td>
<td>Pacific Time)</td>
<td></td>
</tr>
<tr>
<td>Product literature</td>
<td><a href="http://www.altera.com">www.altera.com</a></td>
<td><a href="http://www.altera.com">www.altera.com</a></td>
</tr>
<tr>
<td>Altera literature services</td>
<td><a href="mailto:literature@altera.com">literature@altera.com</a></td>
<td><a href="mailto:literature@altera.com">literature@altera.com</a></td>
</tr>
<tr>
<td>Non-technical customer service</td>
<td>(800) 767-3753</td>
<td>+1 408-544-7000</td>
</tr>
<tr>
<td></td>
<td>(7:30 a.m. to 5:30 p.m. (GMT-8:00) Pacific Time</td>
<td></td>
</tr>
<tr>
<td>FTP site</td>
<td>ftp.altera.com</td>
<td>ftp.altera.com</td>
</tr>
</tbody>
</table>
## Typographical Conventions

This document uses the typographical conventions shown below.

<table>
<thead>
<tr>
<th>Visual Cue</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold Type with Initial Capital Letters</strong></td>
<td>Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: <em>Save As</em> dialog box.</td>
</tr>
<tr>
<td><strong>bold type</strong></td>
<td>External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: <em>f</em>&lt;sub&gt;MAX&lt;/sub&gt;, <code>\qdesigns</code> directory, <em>d:</em> drive, <code>\chiptrip.gdf</code> file.</td>
</tr>
<tr>
<td><strong>Italic Type with Initial Capital Letters</strong></td>
<td>Document titles are shown in italic type with initial capital letters. Example: <em>AN 75: High-Speed Board Design</em>.</td>
</tr>
<tr>
<td><strong>Italic type</strong></td>
<td>Internal timing parameters and variables are shown in italic type. Examples: <em>t</em>&lt;sub&gt;IA&lt;/sub&gt;, <em>n</em> + 1. Variable names are enclosed in angle brackets (&lt; &gt;) and shown in italic type. Example: <code>&lt;file name&gt;</code>, <code>&lt;project name&gt;.pof</code> file.</td>
</tr>
<tr>
<td><strong>Initial Capital Letters</strong></td>
<td>Keyboard keys and menu names are shown with initial capital letters. Examples: <em>Delete</em> key, the <em>Options</em> menu.</td>
</tr>
<tr>
<td><strong>“Subheading Title”</strong></td>
<td>References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: “Typographic Conventions.”</td>
</tr>
<tr>
<td><strong>Courier type</strong></td>
<td>Signal and port names are shown in lowercase Courier type. Examples: <code>data1</code>, <code>tdi</code>, <code>input</code>. Active-low signals are denoted by suffix <em>n</em>, e.g., <code>resetn</code>. Anything that must be typed exactly as it appears is shown in Courier type. For example: <code>c:\qdesigns\tutorial\chiptrip.gdf</code>. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword <code>SUBDESIGN</code>), as well as logic function names (e.g., <code>TRI</code>) are shown in Courier.</td>
</tr>
<tr>
<td>1., 2., 3., and a., b., c., etc.</td>
<td>Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.</td>
</tr>
<tr>
<td>• • •</td>
<td>Bullets are used in a list of items when the sequence of the items is not important.</td>
</tr>
<tr>
<td>✓</td>
<td>The checkmark indicates a procedure that consists of one step only.</td>
</tr>
</tbody>
</table>
## Typographical Conventions

<table>
<thead>
<tr>
<th>Visual Cue</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="hand.png" alt="Hand Pointing" /></td>
<td>The hand points to information that requires special attention.</td>
</tr>
<tr>
<td><img src="caution.png" alt="Caution" /></td>
<td>The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.</td>
</tr>
<tr>
<td><img src="warning.png" alt="Warning" /></td>
<td>The warning indicates information that should be read prior to starting or continuing the procedure or processes.</td>
</tr>
<tr>
<td><img src="arrow.png" alt="Angled Arrow" /></td>
<td>The angled arrow indicates you should press the Enter key.</td>
</tr>
<tr>
<td><img src="feet.png" alt="Feet" /></td>
<td>The feet direct you to more information on a particular topic.</td>
</tr>
</tbody>
</table>
Introduction

This document describes how to use tightly coupled memory in Nios II designs and discusses some possible applications. This document includes a tutorial that guides you through the process of building a Nios II system with tightly coupled memory.

The Nios II architecture provides tightly coupled master ports that provide guaranteed fixed low-latency access to on-chip memory for performance critical applications. Tightly coupled masters can connect to instruction memory and data memory allowing fixed low-latency read access to executable code as well as fixed low-latency read, write, or read and write access to data. Tightly coupled masters are additional instruction or data master ports on the Nios II core, separate from the embedded processor’s instruction and data master ports.

This document assumes you are familiar with the Nios II tightly coupled memory. For details, see the Architecture chapter in the Nios II Processor Reference Handbook.

Reasons for Using Tightly Coupled Memory

A wide variety of performance enhancements can be implemented with these tightly coupled memories.

- Separate exception stack for use only while handling interrupts
- Fast data buffers
- Fast sections of code
  - Fast interrupt handler
  - Critical loop
- Constant access time; guaranteed not to have arbitration delays
- Low resource utilization – If your memory needs are small, an entire program’s code and data can be held in a tightly coupled memory pair.
Trade Offs

There are design trade-offs when balancing the benefits of the more general, averaged over time, speed enhancement provided by a cache, with the specific dedicated memory block consumed by a tightly coupled memory whose sole purpose is to make a single part of the code faster. Software guarantees that performance critical code or data is located in tightly coupled memory. That particular piece of code or data achieves high performance. Locating the code within tightly coupled memory eliminates cache overhead such as cache flushing, loading, or invalidating. Divide on-chip memory equitably to provide the best overall combination of tightly-coupled instruction memory, tightly-coupled data memory, instruction cache, and data cache.

Guidelines for Using Tightly Coupled Memory

In this section guidelines and limitations are identified which you need to know when you design hardware and software with tightly coupled memory.

Hardware Guidelines

The following guidelines apply to Nios II hardware designs that include tightly coupled memory:

- Tightly coupled masters are presented as additional master ports on the CPU.
- An on-chip memory SOPC Builder component is the only memory that can connect to a tightly coupled master port on the Nios II core.
- A tightly coupled master on a processor must connect to exactly one on-chip memory slave port. This slave port cannot be shared by any other master port.
- Each on-chip memory can be connected to at most one tightly coupled master even if it is a dual ported memory.
- Whether data or instruction tightly coupled masters are available depends on the type of Nios II core.
- When using the On-Chip Memory component as a tightly coupled memory for Nios II, you must always create it as a RAM, and not a ROM. Tightly coupled memories configured as ROM will fail.
To conserve logic elements, it is better to have one 2 Kbyte tightly coupled memory, than two tightly coupled memories of size 1 Kbyte.

Figure 1 is a block diagram of a simple Nios II system, which includes tightly coupled memories and other Avalon components.

Software Guidelines

The following two guidelines apply to Nios II software that uses tightly-coupled memory:

1. Software accesses tightly-coupled memory addresses just like any other addresses.

2. Cache operations have no effect when targeting tightly-coupled memory.

Locating Functions in Tightly Coupled Memory

Assigning data to a tightly-coupled data memory also involves using a section attribute. Alternatively, the name of the memory, followed by _BASE is defined as a #define in the system.h and can be used as a pointer to reference the tightly-coupled data memory.
The software example in this tutorial provides a source code example of how to locate a particular source code function in a particular linker section. A function gets declared to reside within a linker section with the C section attribute in the file "timer_interrupt_latency.h". This C header file locates `timer_interrupt_latency_irq()` in the .exceptions section as follows:

```c
extern void timer_interrupt_latency_irq (void* base, alt_u32 id)
__attribute__ ((section (".exceptions")))
```

Linker sections are created by SOPC Builder for each memory module in the system. A source code function can be located within a particular tightly-coupled instruction memory simply by assigning that function to the linker section created for that tightly-coupled instruction memory.

SOPC Builder creates additional linker sections with address mappings that are controlled by SOPC Builder. For the case of the .exceptions section, the physical address offset and memory module to base that linker section in is manipulated through SOPC Builder. The .exceptions section is located in a memory module covered by a tightly-coupled data memory using the Exception Address field of the Processor Function setting found in the Nios II More CPU Settings tab of SOPC Builder.

For additional details on the C section attribute, see the Developing Programs Using the HAL chapter in the Memory Usage section of the Nios II Software Developer’s Handbook.

Tightly Coupled Memory Interface

The term tightly coupled memory interface refers to an Avalon-like interface that connects one master to one slave. See Figure 1. Tightly coupled memory interfaces connect tightly coupled masters to their tightly coupled slaves. Tightly coupled memory interfaces are designed to be connected to one port of an on-chip memory device. These devices are known as altsyncrams to Verilog/VHDL designers.

Restrictions

There are a few restrictions which must be observed when designing with tightly coupled memories:

- Tightly coupled slaves must be on-chip memories.
Only one master and one slave can be connected to a given tightly coupled memory interface, which makes the tightly coupled memory interface a point-to-point connection.

- Tightly coupled slaves have a data width of 32 bits. Tightly coupled memory interfaces do not support dynamic bus sizing.
- Tightly coupled slaves have a read latency of 1 cycle, a write latency of 0 cycles, and no wait states.

When tightly coupled memory is present, the Nios II core decodes addresses internally to determine if requested instructions or data reside in tightly coupled memory. If the address resides in tightly coupled memory, the Nios II core accesses the instruction or data through the tightly coupled memory interface. Accessing tightly coupled memory bypasses cache memory. The processor core functions as if cache were not present for the address span of the tightly coupled memory. Instructions for managing the cache do not affect the tightly coupled memory, even if the instruction specifies an address in the range occupied by a tightly coupled memory.

**Dual Port Memories**

Each tightly coupled master connects to one tightly coupled slave over a tightly coupled interface. For this reason, it is helpful to use dual port memories with the tightly coupled instruction master (see Figure 1). The tightly-coupled instruction master is incapable of performing writes, since it is designed to only access code for execution. Without a second memory port connected to an Avalon data master, the system is denied write access to the tightly-coupled instruction memory. Without write access, code cannot be downloaded into the tightly coupled memory by the Nios II IDE which makes development and debugging difficult. Without a second port on the tightly-coupled instruction memory, no data master has access to the memory, meaning the user has no way of viewing the contents. By making the tightly-coupled instruction memory dual port the embedded processor’s data master can be connected to the second port, allowing both reading and writing of data.

**Building a Nios II System with Tightly Coupled Memory**

This section gives a detailed list of instructions for creating a Nios II system in SOPC Builder that uses two tightly-coupled memories, one instruction and one data. These two tightly coupled memories will
be connected to the Nios II processor as shown in Figure 1. Additionally, instructions will be listed to build a software project to exercise these tightly coupled memories. The output of the software will show that the tightly coupled memories are much faster to access than other on-chip memories.

In this section you will perform the following steps:

1. Modify an existing reference design to include tightly coupled memories.

2. Create the tightly coupled memories in SOPC Builder.

3. Connect the tightly coupled memories to masters.

4. Position the tightly coupled memories in the Nios II processor's address map.

5. Specify the Nios II exception address to access tightly-coupled instruction memory.

6. Add a performance counter.

7. Generate the hardware system.

8. Create a software project to exercise tightly coupled memories.

9. Specify a separate exception stack.

10. Change the projects' build properties.

11. Execute the software on the new hardware design.

**Hardware and Software Requirements**

The following hardware and software are required to perform this exercise:

- Nios II development tools version 5.0
- Quartus II software version 5.0
- Any Nios development board
Modify the Example Design to Include Tightly Coupled Memories

First, you will create a new hardware reference design with tightly coupled memories that is based on any of the standard reference design installed with the Nios II development tools. To create this modified reference design, perform the following steps:

1. In your host computer file system, locate the standard design directory for your chosen development board and HDL. For example, on a Windows PC C:\altera\kits\nios2\examples\verilog\niosII_cycloneII_2c35 \standard contains the Verilog HDL design files for the Nios Development Board, Cyclone II Edition.

2. Copy the standard directory to a new directory named standard_tcm.

3. Open the standard.qpf design project in Quartus II version 5.0.

4. On the Tools menu, click SOPC Builder.

5. Double click the cpu component from the list of available components on the System Contents page to open the Altera Nios II Processor configuration wizard.

Figure 2. Selection of the Nios II/f Core in the Nios II Processor Wizard
6. On the Nios II Core tab, select Nios II/f. See Figure 2.

7. Click the Caches and Tightly Coupled Memories tab.

8. Select Include tightly coupled instruction master ports.

9. Select Include tightly coupled data master ports.

10. Verify the default settings that are shown in Figure 3.

11. Click Finish to close the Nios II Processor configuration wizard.
Two new master ports now appear under the cpu component called tightly_coupled_data_master_0 and tightly_coupled_instruction_master_0. See Figure 4. These master ports are not yet connected to slave ports.

Create the Tightly Coupled Memories

Two types of tightly coupled memories will be created: a tightly-coupled instruction memory and a tightly-coupled data memory.

1. In the left-hand pool of components, scroll down and double-click Memory to expand the list of available memory components.

2. In the list of available memory components, double-click On-Chip Memory. The On-Chip Memory configuration wizard appears. See Figure 5.
3. Under the Memory Type setting, check Dual-Port Access to configure the memory as dual port.

4. Configure the settings that are listed in Table 1 and shown in Figure 5.
Table 1. On-Chip Memory Default Settings

<table>
<thead>
<tr>
<th>Properties</th>
<th>Configuration Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Type</td>
<td>RAM/ROM</td>
</tr>
<tr>
<td>RAM writeable</td>
<td></td>
</tr>
<tr>
<td>Dual-Port Access</td>
<td>Select</td>
</tr>
<tr>
<td>Block Type</td>
<td>Automatic</td>
</tr>
<tr>
<td>Size</td>
<td>Memory Width</td>
</tr>
<tr>
<td>32 Bits</td>
<td></td>
</tr>
<tr>
<td>Total Memory Size</td>
<td>Total Memory Size</td>
</tr>
<tr>
<td>4 Kbytes</td>
<td></td>
</tr>
<tr>
<td>Read Latency</td>
<td>Slave s1</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Slave s2</td>
<td>1</td>
</tr>
</tbody>
</table>

5. Click **Finish** to close the On-Chip Memory configuration wizard.

6. Click the **System Contents** tab, scroll down, and then double-click the **onchip_memory_0** component.

   ⚠️ In this tutorial, you must name the components exactly. If your component names differ from the names printed here, the software example will not work.

7. Right-click **onchip_memory_0** and rename the component to **tightly_coupled_instruction_memory**.

8. In the list of available memory components, double-click **On-Chip Memory** to open the On-Chip Memory configuration wizard. See **Figure 6**.
9. Configure the settings that are listed in Table 2 and shown in Figure 6. Unlike tightly coupled instruction memory, this memory will be single-port so you will NOT check Dual-Port Access. Total memory size for tightly-coupled data memory will be twice the size of tightly-coupled instruction memory at 8 Kbytes.
Using Nios II Tightly Coupled Memory Tutorial

<table>
<thead>
<tr>
<th>Table 2. On-Chip Memory Default Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
</tr>
<tr>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>Memory Type</td>
</tr>
<tr>
<td>RAM (writeable)</td>
</tr>
<tr>
<td>Dual-Port Access</td>
</tr>
<tr>
<td>Block Type</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>32 Bits</td>
</tr>
<tr>
<td>Total Memory Size</td>
</tr>
<tr>
<td>Read Latency</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>Slave s2</td>
</tr>
</tbody>
</table>

10. Click Finish to close the On-Chip Memory configuration wizard.

11. Rename the onchip_memory_0 component to tightly_coupled_data_memory.

Connect & Position the Tightly Coupled Memories

Now that the tightly coupled memories have been created, they must be associated with their masters. To associate masters with the tightly coupled memories, perform the following steps:

1. Click each new tightly coupled memory and click Move Up to move the individual memories just below the cpu component. This will facilitate creating the patch-panel connections between components.

2. Click the "+" to expand the tightly_coupled_instruction_memory component.

3. Using the patch-panel connection matrix in SOPC Builder, disconnect the s1 port of tightly_coupled_instruction_memory from all masters except for the tightly_coupled_instruction_master_0 component listed under the cpu component.

   The remaining connection becomes the tightly-coupled memory interface shown in Figure 1 for the tightly-coupled instruction memory, connecting the tightly-coupled instruction master port to the tightly-coupled slave port on the tightly-coupled instruction memory.
The sides of the Module Name column label must be widened to view the entire module name, including the slave port numbers. To widen the view, click and drag the outward edges of the Module Name title box column.

4. Similarly, disconnect the s2 port of the tightly_coupled_instruction_memory from all masters except the embedded processor’s data_master.

This connection is shown in Figure 1 as the Avalon switch fabric connection between the Avalon data master port and the Avalon slave port on the tightly-coupled instruction memory. At this time of connection, port s2 of this dual-ported memory becomes an Avalon slave port, not a tightly-coupled slave port. This is because s2 is now connected to an Avalon master, not a tightly coupled master.

5. Click the “+” to expand the tightly_coupled_data_memory component.

6. Disconnect the s1 port of tightly_coupled_data_memory from all masters except tightly_coupled_data_master_0.

7. The remaining connection becomes the tightly-coupled memory interface shown in Figure 1 for the tightly-coupled data memory, connecting the tightly-coupled data master port to the tightly-coupled slave port on the tightly-coupled data memory.

8. Click the Base address for s1 and change the value to 0x04000000.

9. The End address will also be updated to reflect the size, and should now hold the value 0x04000FFF.

10. Click the Base address for s2, and change its value to 0x04000000.

11. Verify that the End address for s2 changes to 0x04000FFF.

Base address selection is important. Tightly coupled memories must be mapped so that their addresses DO NOT overlap with the embedded processor’s memories and peripherals that are connected to its Avalon instruction and data masters.
Tightly coupled memories should be mapped to an address with unique upper address bits (ideally just one bit) to simplify address decoding. These connections are intended to be fast, so if the address decoding logic gets too complicated, \( F_{\text{max}} \) will be compromised. The Nios II component will work correctly if it detects sub-optimal address decoding and will display a warning during generation detailing the detected overlapping addresses.

As an example of optimal address mapping, if all the normal memories and peripherals in your system occupy addresses below 0x2000000, mapping your tightly coupled memories at addresses from 0x2000000 and above will satisfy this requirement.

12. Click the Base address for \( s1 \) and change the value to 0x04004000. The End address will also be updated to reflect the size, and should now hold the value 0x04005FFF.

When adding more tightly coupled memories, minimize the address decoding that takes place by assigning a unique address bit to the tightly coupled memories to separate them from the other components. This minimizes the amount of address decoding logic required, and will maximize speed when accessing tightly coupled memories.

### Exception Address Selection

In this section you will position the exception address within a tightly coupled memory. To position the exception address, perform the following steps:

1. Click the Nios II More “cpu” Settings tab in SOPC Builder. See Figure 7.

2. Set the exception address memory module to tightly_coupled_instruction_memory/s1.

The Address fields are populated automatically, indexed off of the base address specified for the memory module on the System Contents tab.

![The sides of the Memory Module column label must be widened to view the entire table of module names, including the slave port numbers. To widen the view, click...](image-url)
Add a Performance Counter

You will want a mechanism to determine how fast reading and writing to tightly coupled memories can be done by Nios II as compared to other memories. A performance counter peripheral is perfect for this task.

1. Click the System Contents tab.

2. In the left-hand pool of components, double-click Extra Utilities to expand the list of available components.

3. In the list of available components, double-click Performance Counter Unit to open the Performance Counter configuration wizard.

4. Click Finish, accepting the default settings of the three simultaneously-measured sections.
5. Rename the `performance_counter_0` component to `performance_counter`.

6. Verify that the SOPC Builder system contains the components with the same patch-panel connections as shown in Figure 8.

**Figure 8. Final SOPC Builder memory module configuration**

Build the Hardware System

Now the hardware system is ready for generation and compilation. To generate and compile the hardware system, perform the following steps:

1. Click **Generate**.

2. When generation is done, click **Exit**.

3. In the Quartus II software, on the **Processing** menu, click **Start Compilation**.
4. When the Quartus II software compilation is finished, on the Tools menu click Programmer to program the newly generated standard.sof into the FPGA.

Create a Software Project to Exercise Tightly Coupled Memories

Now we’ll need a software example to exercise the tightly coupled memories on this new hardware design. Perform the following steps:

1. On the Tools menu, click SOPC Builder.

2. Click the System Generation tab.

3. Click Run Nios II IDE.

4. Create a new software project called tcm_vs_cache based on the Tightly Coupled Memories template.
   a) On the File menu, point to New and click Project.
   b) In the New project dialog box, select C/C++ Application and click Next.
   c) In the Name field, type tcm_vs_cache.

5. Copy the C source file tcm_vs_cache.c into your project’s source directory. The tcm_vs_cache.c file can be found on the Nios II Literature web page with this tutorial.

6. Delete the source file tcm.c in your project’s source directory.

Specify a Separate Exception Stack

Although the impact of the separate exception stack setting is not demonstrated in this software example, it is pointed out here because it will benefit most designs that leverage tightly coupled data memories. An interrupt speed-up is not seen in this example because the computation measured occurs without any interrupts having a chance to fire.
1. Right-click the \texttt{tcm_vs_cache_syslib} project and select \textbf{Properties}.

2. Click \textbf{System Library}.

3. Check \textbf{Use a separate exception stack} as shown in \textbf{Figure 9}.

\textbf{Figure 9. Nios II IDE Memory Module Specification for Separate Exception Stack}

4. Check \textbf{Use a separate exception stack} as shown in \textbf{Figure 9}.

5. Verify that the default \textbf{Maximum exception stack size (bytes)} is \textbf{0x400} bytes.

6. From \textbf{Exception stack memory}, select \textbf{tightly_coupled_data_memory}.

7. Click \textbf{Apply} and then click \textbf{OK}.

In the Windows pull-down menu, select \textbf{Preferences}.

Under the \textbf{C/C++ Editor} from the \textbf{Appearance} tab, check \textbf{Show line numbers}.

8. Click \textbf{Apply} and then click \textbf{OK}.
Change the Projects’ Build Properties

You will change the project build properties for both the application and system projects to Release Mode. This will set the compiler optimization flags for both projects to –O2. To change the project build properties, perform the following:

1. Click the tcm_vs_cache project.
2. Right-click tcm_vs_cache and then select Properties.
3. Select C/C++ Build.
4. Select Release from the Configuration list of options.
5. Click Apply and click OK.
6. Select the tcm_vs_cache_syslib project to highlight it.
7. Right-click tcm_vs_cache_syslib and then select Properties.
8. Select C/C++ Build.
9. Select Release from the Configuration list of options.
10. Click Apply and click OK.
11. To build the tcm_vs_cache project, select the tcm_vs_cache project.
12. Right-click tcm_vs_cache and then select Build Project.

Executing the Software on the Modified Standard_tcm Hardware Design

In this section you will download the program to the board and analyze the performance results.
Download and Run Tcm_vs_cache Project

Now you are ready to witness the results of your efforts in this tutorial.

✔ Download and Run the `tcm_vs_cache` project.

Observe the statistics which illustrate the higher speeds obtained by leveraging tightly coupled memories. The output is similar to the following collected on a 1s40 Nios development board. The timing numbers output varies between Nios development boards.

Tightly Coupled Memory vs. Cache Example Real-Time Measurements:
Altera Nios development board FPGA type: Stratix (1S40).
Interrupt response time: 79 clock cycles.

Checksum Times: All 5 memories match.
Tightly coupled memory: 18.20 microseconds, 910 clocks.
On-chip memory: 27.48 microseconds, 1374 clocks.
On-chip memory (cached): 18.26 microseconds, 913 clocks.
SDRAM memory: 32.80 microseconds, 1640 clocks.
SDRAM memory (cached): 18.16 microseconds, 908 clocks.

Checksum loop measured iteration: 3.
Checksum value: 34465 total.
Checksum block size: 320 words (32 bits).
CPU Frequency: 50.0000 Mhz.

The output of the program is the result of samples from two types of measurements; interrupt response time and memory access times, which are described in the following sections.

Interrupt Response Time

The interrupt service routine (ISR) for the high resolution timer is used to measure elapsed time between firing and handling of an interrupt. The high resolution timer's snapshot register is read when the ISR is entered. The timer has been decrementing since the actual
firing of the hardware timer interrupt occurred, at which point the timer snapshot register was loaded with the period value.

Therefore, by subtracting the value of the snapshot register read in the ISR from the high resolution timer’s initialized period value, the actual number of clock ticks which have elapsed between the firing of the interrupt and the entry to the ISR can be determined.

This number of clock cycles is reported as the interrupt response time, also commonly referred to as the interrupt latency.

Memory Access Times

Memory access times are measured by leveraging a simple checksum algorithm. First, a repeating pattern (0-255) of numbers is written to each memory type. The memory types include Tightly Coupled Memories, SDRAM, and for Nios development boards that have 64 Kbyte On-Chip RAM. Each checksum measurement uses the same checksum block size.

Each checksum value is compared to the others to verify that they are the same, indicating that no errors occurred while accessing individual memory addresses. The checksum for each memory type is calculated twice in a row. Before the calculation is computed the first time for each memory type, the cache is flushed. This ensures that every data access will be a cache miss. The second time, the calculation is listed as a ‘cache’ value, since the percentage chance of a data cache hit on the second calculation round is greatly increased. There is not a ‘cache’ value calculated for the tightly coupled memory, because the tightly coupled memory is directly connected to the Nios II core, bypassing any data cache memory. In fact, accessing a tightly coupled memory is as fast as accessing cache memory.

Software Example Implementation Notes

Checksum Memory Block Size

The macro CHECKSUM_MEMORY_BLOCK_SIZE defines the size of each memory block used for measuring the performance of each of the tightly coupled memory (M4K), on-chip memory (M-RAM), and off-chip memory S-DRAM. The value of the CHECKSUM_MEMORY_BLOCK_SIZE can be modified to observe the effects of varied block sizes on data cache hits and misses and how
that compares to tightly coupled memories. Take care that the size of the block is not defined to be so large that it overlaps the exception stack defined at the top of the tightly-coupled data memory.

The chance of a data cache miss can be greatly increased with the selection of particular block sizes which, when combined with particular data cache line sizes, cause cache tag bit values resulting in a worst case cache thrashing situation.

**Heating the Instruction Cache**

The checksum loop is performed multiple times to heat up the instruction cache. This prevents any instruction cache misses from impacting measurement results we are interested in, which are strictly the data memory access times.

The number of loop iterations performed before the checksum performance time is measured is controlled by the macro CHECKSUM_LOOP_MEASURED_ITERATION. If this number is increased to a large value, it is possible that the impact of a timer interrupt could skew a single calculation.

**Unrolled Checksum Loops for Optimization**

The checksum calculation loops have been unrolled to maximize parallelization during data read latency periods. Memory reads are done four at a time. Then a sum is performed. This ordering of four reads followed by four summations streamlines data reads on the Nios II Avalon data master port or tightly coupled data master port. The first data element has some amount of read latency associated with the data access, so the read instructions are issued to start reading the other three data elements because the summation instruction cannot operate on the first data element until the data value is retrieved from memory. The other data element values are retrieved from memory in parallel during the summation of the first data element values.

**Conclusion**

With careful planning, you can achieve large performance gains by leveraging tightly coupled memory. Use of a separate exception stack with a tightly-coupled data memory, along with assignment of
the `.exceptions` section to a tightly-coupled instruction memory, will benefit almost every embedded application. Consider adding most interrupt service routines to the `.exceptions` section. Use a tightly-coupled instruction memory large enough to hold all of the desired functions. Additional empirical analysis which compares various cache sizes against dedicating portions of code to tightly coupled instruction memories can identify application specific combinations which will net large performance gains for embedded Nios II applications.