



Allocating Address Space in a 32-bit Processor

Summary

This application note provides detailed information on mapping memory and peripherals into a 32-bit processor's address space.

Application Note
AP0149 (v2.0) February 27, 2008

An FPGA design incorporating a 32-bit processor will typically involve the connection of slave memory and peripheral devices – to the processor's External Memory and Peripheral I/O interfaces respectively. This is physical connection, the wiring of the devices to the processor. At this point, the processor – and more importantly the embedded code therein – is 'unaware' of the existence of such devices, with respect to its address space. These physical slave devices must now be mapped into specific locations within that address space.

The 'memory map', as it is often called, is essentially the bridge between the hardware and software projects – the hardware team allocating each of the various memory and peripheral devices their own chunk of the processor's address space, the software team then writing their code to access the memory and peripherals at the given locations.

Address Space Mapping - Overview

Processor address space can be configured from within the FPGA project (both Memory and Peripheral I/O) or the Embedded Software project (Memory only). In both cases, mapping can be achieved quickly by importing slave device information directly from the relevant schematic sheet. Once imported, the mapping of devices can be 'hand-crafted' as required. With respect to the Embedded Software project, you can also opt to automatically import dedicated memory mapping definitions directly from the FPGA project upon compilation.

You can also generate `.asm` and/or `.h` files for the Embedded Software project, upon FPGA project compilation, providing a means to quickly and efficiently address devices from within the embedded source code.

Figure 1 summarizes the key elements to defining processor address space in terms of memory and peripheral I/O and where such elements can be defined.

Allocating Address Space in a 32-bit Processor

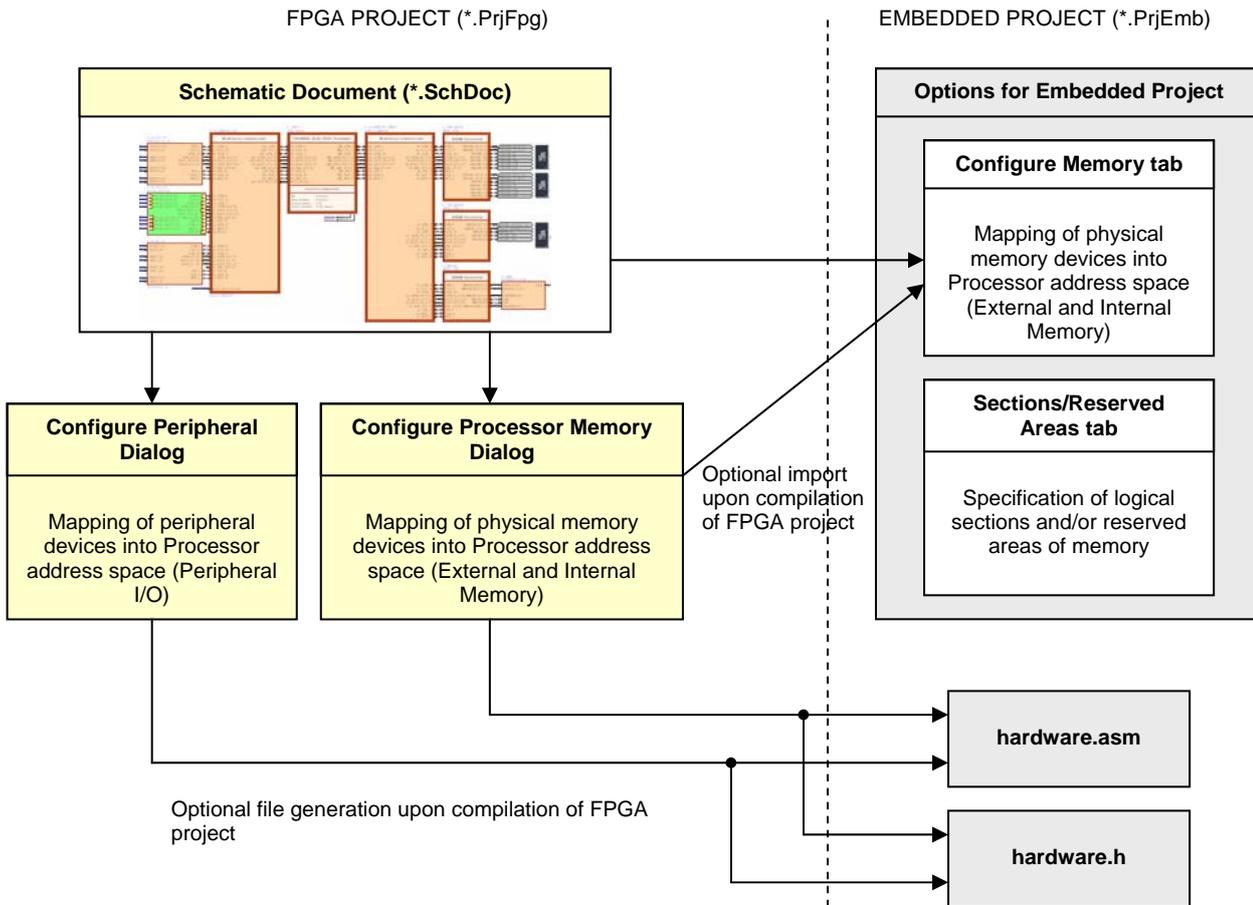


Figure 1. Configuring Processor Memory and Peripheral I/O.

A Word on Mapping Physical Memory...

When configuring processor memory, the physical memory devices are mapped into the processor's address space – in the Internal and External Memory ranges – but can also be further broken down into separate logical spaces (or sections) for reference by and within the embedded source code. Figure 2 illustrates this concept of memory space allocation.

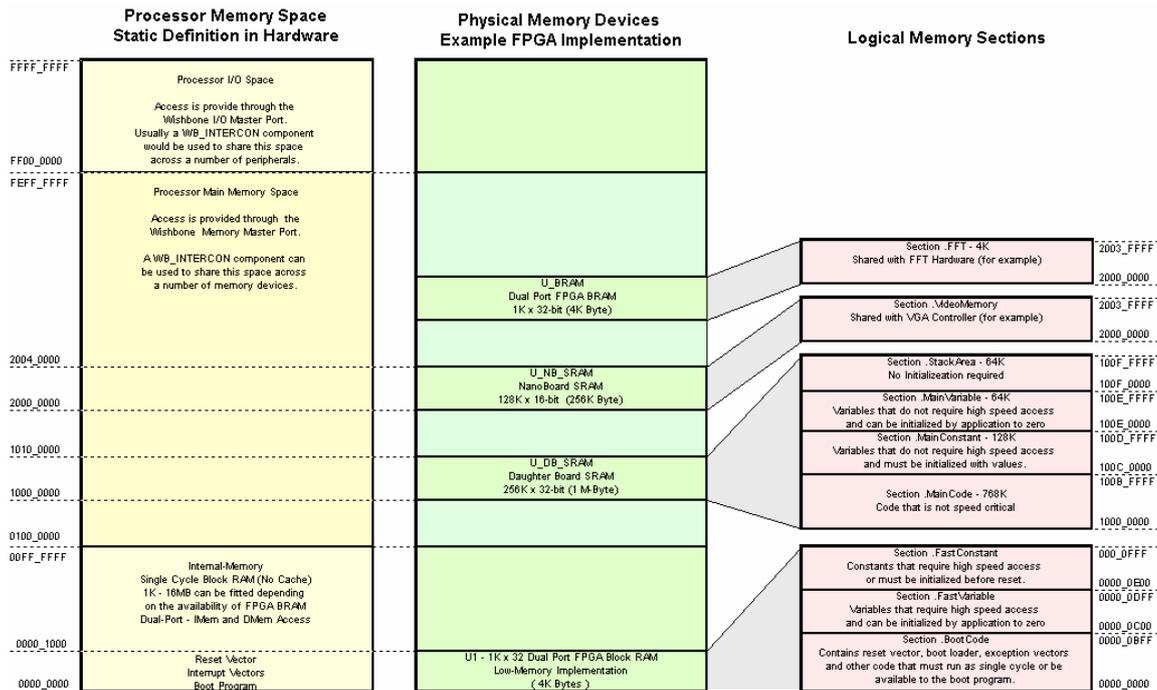


Figure 2. Allocating processor and physical device memory spaces.

From left to right, Figure 2 shows:

- The predefined memory architecture of the 32-bit processor. The 4GB linear address space divided between Internal Memory (16MB), External Memory and Peripheral I/O space (16MB). This architecture is static (i.e. cannot be changed by the user)
- Physical memory spaces. In the example of Figure 2, there are four physical memory spaces – Internal Memory, Daughter Board SRAM, NanoBoard SRAM and FPGA BRAM. The dotted lines indicate where in the processor’s address space these physical devices are mapped, and the sizing of each
- Logical memory spaces, showing the mapping of specific sections within the embedded code to the physical device memory.

Mapping Physical Memory from within the FPGA Project

Typically, the required physical memory devices will be added to the design and wired to the processor’s External Memory interface accordingly, before mapping is performed. To illustrate, consider the circuitry of Figure 3.

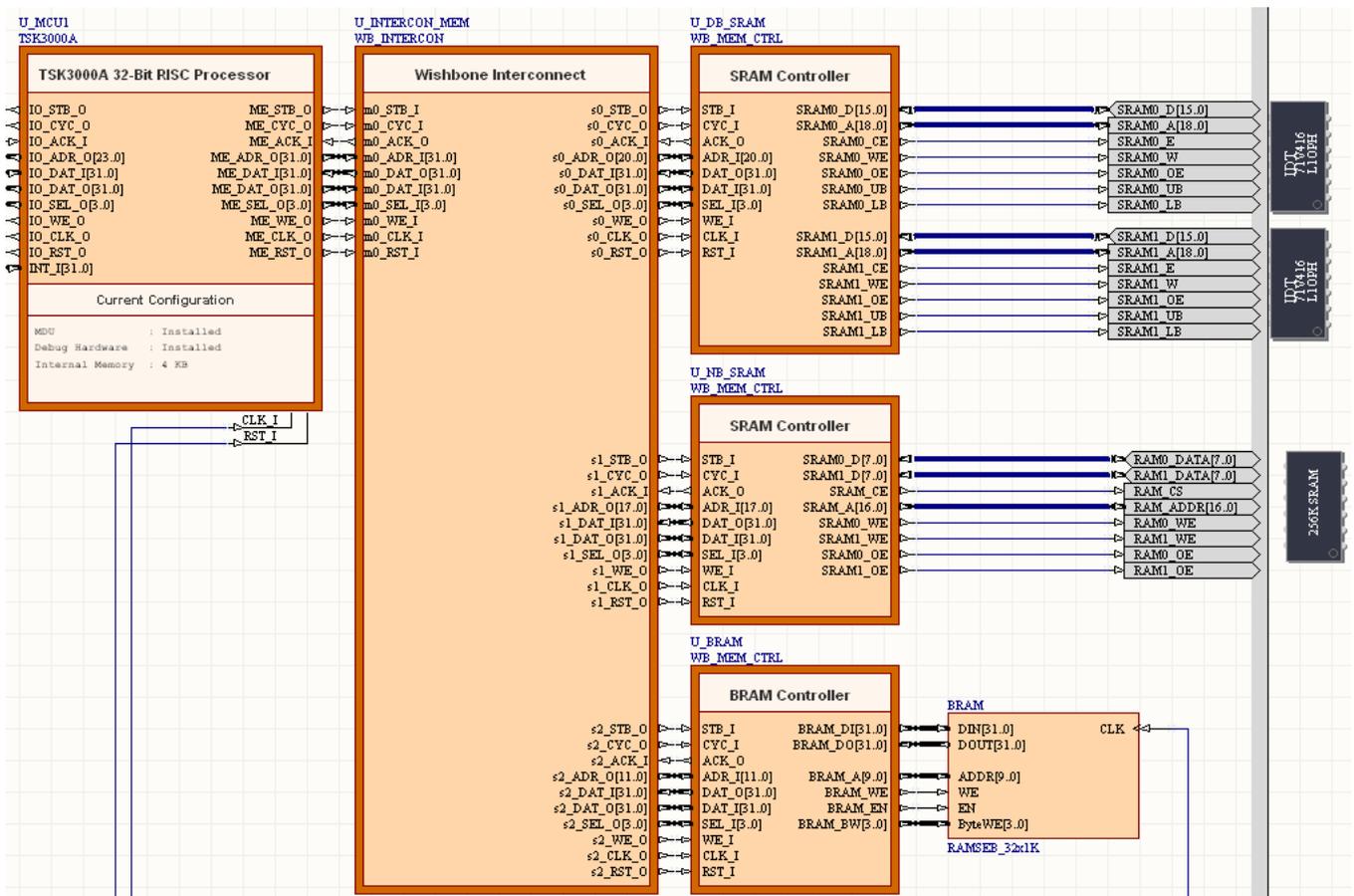


Figure 3. Connection of physical memory devices to the 32-bit processor.

Here, physical memory on the NanoBoard and Daughter Board, as well as Block RAM within the target FPGA device, is connected to the processor’s External Memory interface. The processor in this case is a TSK3000A, but could just as easily be swapped for any of the other 32-bit processors available in Altium Designer. Connection to the interface is made through use of a configurable Wishbone Interconnect device (WB_INTERCON).

For more information on connecting physical memory devices to a processor, refer to the application note [Connecting Memory and Peripheral Devices to a 32-bit Processor](#).

From within the FPGA project, mapping of physical memory into the processor’s address space is performed using the *Configure Processor Memory* dialog (Figure 4). Access this dialog by right-clicking on the symbol for the processor (in the schematic) and choosing **Configure Processor Memory** from the context menu that appears. Alternatively, use the **Configure Processor Memory** button available from the *Component Properties* dialog for the processor.

Allocating Address Space in a 32-bit Processor

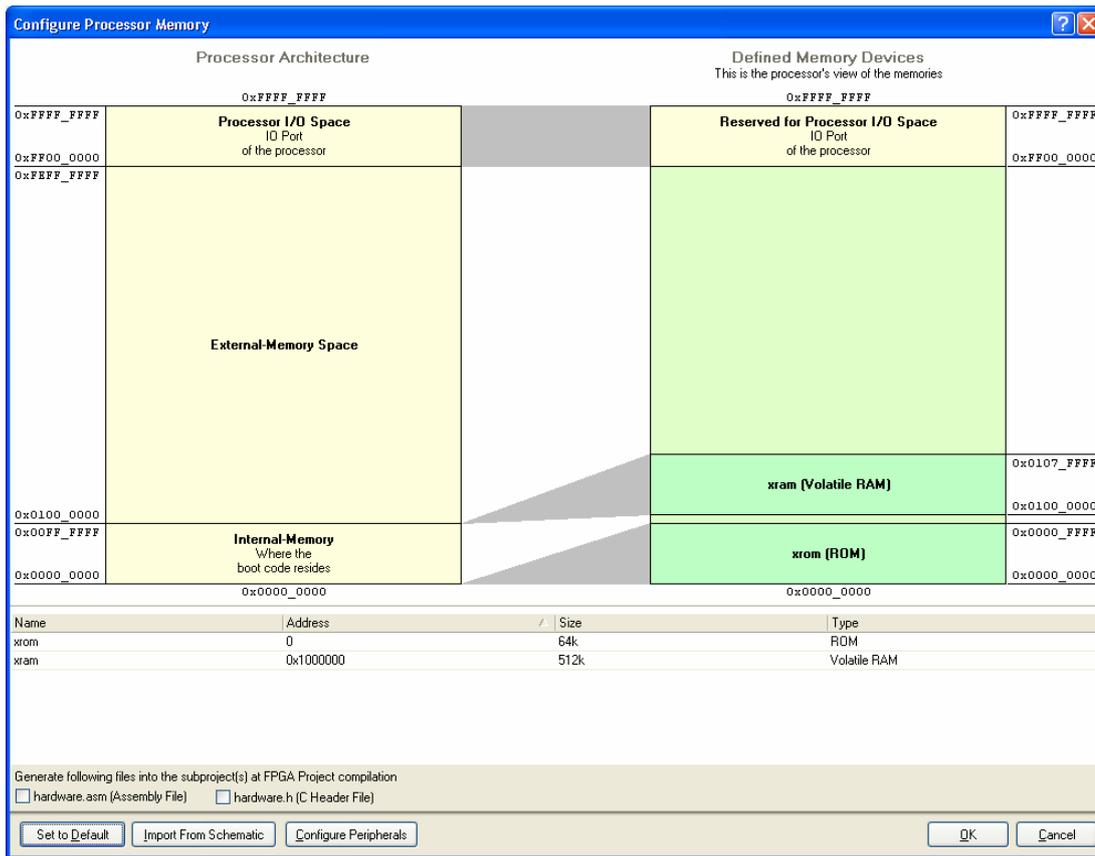


Figure 4. Configuring processor memory from within the FPGA project

The dialog is divided into two regions. The top region provides a visual summary of the memory devices that are currently defined and mapped into the processor's address space – i.e. the processor's 'view' of the actual physical memories in the design. The bottom region of the dialog provides a tabular listing of each of these defined memory devices.

Default memory devices are initially defined and mapped. These will vary depending on the processor type being configured. For the TSK3000A in our example circuit, the following devices are mapped by default:

- **xrom** - this ROM device is mapped into the processor's Internal Memory address range, in order to cater for storage of the processor's embedded code. Starting at address 0000_0000h, it is sized to give 64KB of Internal Memory
- **xram** - this RAM device is mapped into the processor's External Memory address range in order to cater for the storage of data. Starting at address 0100_0000h, it is sized to give 512KB of external memory.

Bear in mind that these are default memory mappings – the actual physical devices may not yet exist in the design or, if they do, may not be identical in size.

Controls are also provided in the dialog for generating header files (Assembly or C) upon compilation of the FPGA project.

The dialog also provides a **Configure Peripherals** button – allowing you to quickly jump between memory and peripheral configuration dialogs. Note that when using this facility, you will be prompted to confirm whether you wish to save the configuration of the current dialog before proceeding to the other dialog.

Importing Device Information from the FPGA Project Schematic

If you have connected your physical memory device(s) to the processor through a Wishbone Interconnect, then the simplest way to make these devices 'known' to the processor – and automatically have them mapped into the processor's address space – is to use the *Configure Processor Memory* dialog's *Import from Schematic* feature.

Launched by clicking on the **Import From Schematic** button, the feature essentially scans the FPGA design for configurable Wishbone Interconnect devices. All detected interconnect devices, along with the slave devices associated to them, are presented in the *Choose Wishbone Items* dialog (Figure 5).

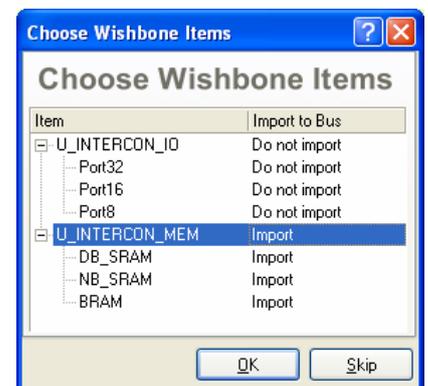


Figure 5. Selecting memory devices to import into processor address space.

Prior to the dialog appearing, you will be given the choice of whether or not to delete any existing memory device definitions that have been added. If you have not yet configured the processor memory, or want to start over again from a 'blank canvas' as it were, then opting to delete here is the best course of action.

On the other hand, you may have already defined memories that you wish to keep. This can be the case when adding additional physical memory to an already well-established design. In this case, it is better to not to delete the existing definitions, but rather import just the addition(s).

Each slave memory device connected to an Interconnect is listed in the dialog, in terms of its identifier. This identifier is assigned as part of the slave device's definition when configuring the Interconnect. The dialog offers the ability to import selected devices or all devices associated to a given Interconnect, by clicking on the relevant entry in the **Import to Bus** column. Choose the physical memory devices you wish to import and click **OK**.

Importing will:

- Add memory device definitions where they currently do not exist. The identifier of an Interconnect slave device – defined when configuring the Interconnect – will be used to name the memory device definition.
- Update any existing memory device definitions, only if the identifier of the Interconnect slave device matches that of an existing memory device definition.
- Preserve existing memory device definitions whose identifiers do not match those of the Interconnect slave devices being imported – leaving hand-crafted definitions untouched.

Update to/preservation of existing memory device definitions, will only be possible where you have opted to not delete the existing configuration before importing.

 Only the internal memory defined for the processor and physical memory devices connected to the processor via a Wishbone Interconnect device will be imported using the Import from Schematic feature. If a memory device is connected directly to the processor's External Memory interface (via the appropriately configured memory controller and any additional wiring), it will not be imported and mapped.

Figure 6 illustrates an example of this mapping based on the example schematic circuitry of Figure 3. The default memory definitions have been deleted prior to import.

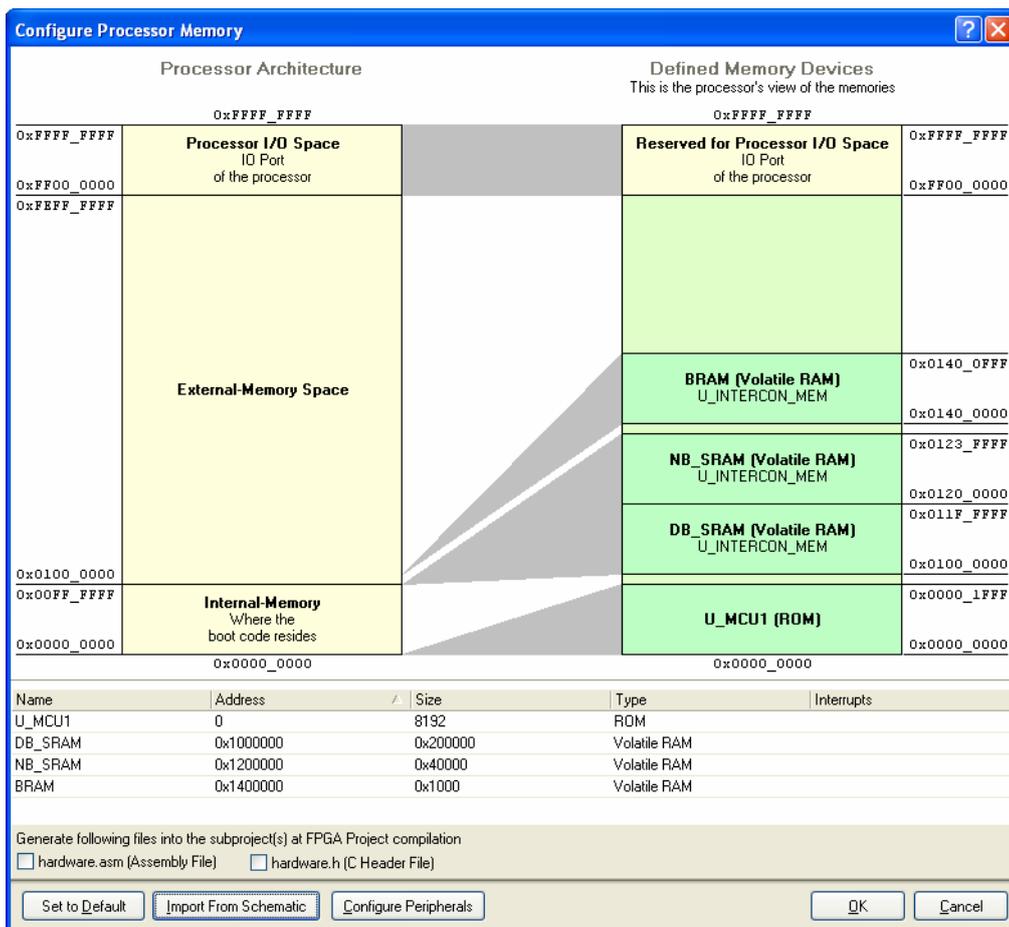


Figure 6. Defining memory devices automatically by importing from the schematic.

If you do not wish to import any devices, simply click on the **Skip** button (in the *Choose Wishbone Items* dialog) – only the definition of the processor's internal memory will be imported.

Managing Memory Device Definitions

Importing from the FPGA project schematic will define memories from the processor's perspective that are exactly in-line with the physical memory devices placed and wired-up on the schematic sheet. Once imported, you may well want to hand-craft how the processor really 'sees' these memories. For example, you may want to specify a single physical memory device in the design to be seen as several memory devices by the processor – essentially 'carving up' the address range of the true physical device into sub-ranges.

New memory device definitions may be added, or existing ones modified or deleted, using the available commands on the right-click menu for the *Configure Processor Memory* dialog. An existing memory device can also be modified with respect to its definition by double-clicking on its corresponding entry in the list section of the dialog, or by double-clicking on its 'block' in the graphical display. The associated *Processor Memory Definition* dialog will appear (Figure 7).

The dialog enables you to accurately define the memory device in terms of its size and base address – specifying where in the processor's address space it is to be mapped. You can specify these values using decimal or Hex notation (e.g. 10000, 0x10000, 1k, 64k, 1M).

The **Name** field allows you to define a unique identifier for the device. The identifier for each memory device will be used when generating header files for inclusion into the Embedded Software project (see the section [Generating Header Files for the Embedded Software Project](#)). The identifiers will also be used to uniquely identify the corresponding output HEX files.

The identifier used must not contain any spaces.

The **Type** region of the dialog allows you to set the type and relative speed of the memory device. The memory type can be either ROM or RAM (volatile or non-volatile). Six speed settings are available to choose from, ranging from 0 (fastest) to 5 (slowest). The Linker uses the speed settings for all defined memories in order to best optimize the overall performance and efficiency of the code.

How the processor's memory address space is allocated to the various physical memory devices in a design ultimately depends on the function of the physical memory device and individual design requirements. All but one physical memory device will be mapped into the processor's External Memory address range. The exception is the Dual port FPGA Block RAM used to implement the processor's internal memory. This will be mapped into the processor's Internal Memory address range. This will always be mapped starting at the base address of 0000_0000h. The processor's Internal Memory address range is 16MB. The actual size of the physical RAM mapped into this range will be driven by how much physical BRAM is available in the target FPGA device. This is typically between 4KB and 1MB.

The memory device definitions are stored as part of the processor component.

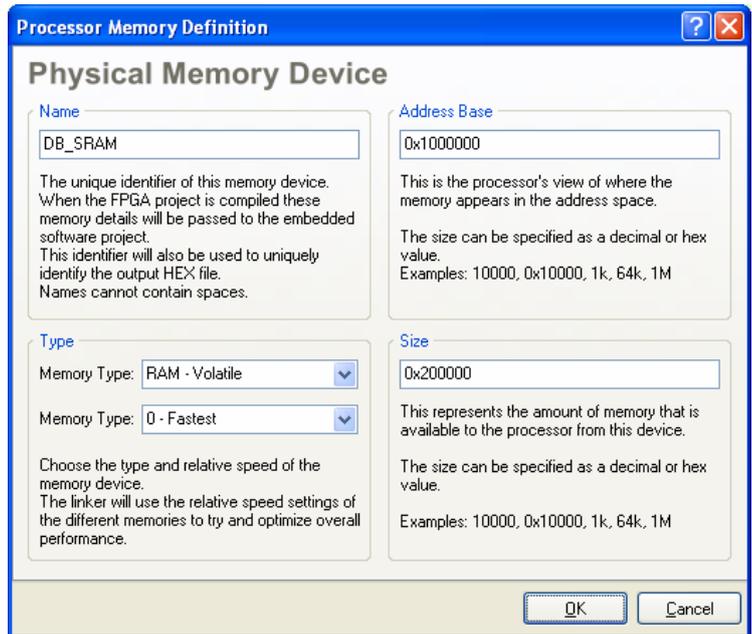


Figure 7. Defining a mapped physical memory device.

Mapping Physical Memory from within the Embedded Project

From within the Embedded Software project, mapping of physical memory into the processor's address space is performed from the **Configure Memory** tab of the *Options for Embedded Project* dialog (Figure 8). Access this dialog by right-clicking on the project's entry in the *Projects* panel and choosing **Project Options** from the context menu that appears.

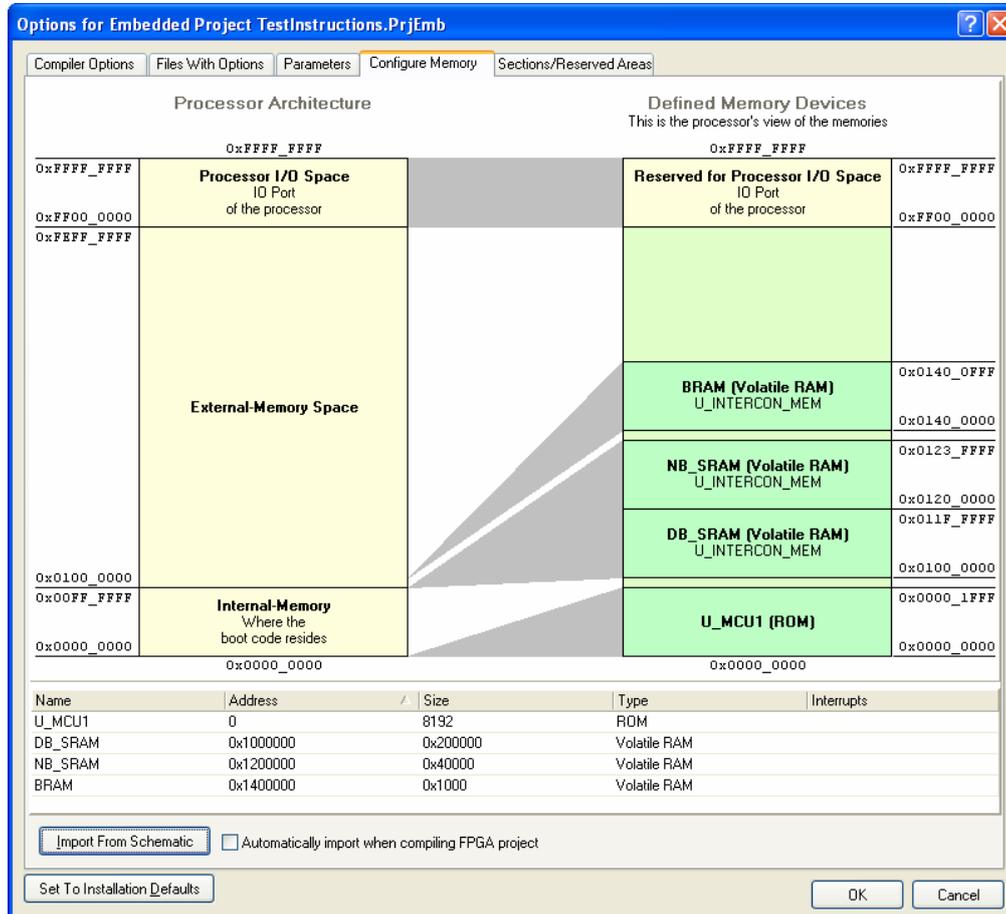


Figure 8. Configuring processor memory from within the Embedded Software project

Controls for memory device definition are similar to those found in the *Configure Processor Memory* dialog when configuring memory within the FPGA project. Use the right-click menu to access commands for adding new memory definitions, or for editing/deleting existing definitions.

It is quite common for the Embedded software to be developed in parallel with the FPGA design and, with the two not linked, the Embedded Software Developer has no knowledge of the physical memory devices being placed by the FPGA Designer. Typically, the Embedded Software Developer will make an educated guess as to the memories available and continue to develop accordingly, and independently.

When the time comes to synchronize the two independently-developed projects – linking the Embedded project as a sub-project of the FPGA project – the Embedded Software Developer simply imports the memory definitions for the processor, from the FPGA design. This can be achieved in two ways:

- **Manually** – by using the *Import From Schematic* button. The memory definitions stored with the processor component are imported directly from the FPGA design, provided that the design has been compiled.
- **Automatically** – by enabling the *Automatically import when compiling FPGA project* option. When the FPGA project is compiled, the memory device definitions for the processor will automatically be passed to the linked embedded software project.

 If you are manually defining memory device definitions in the **Configure Memory** tab of the *Options for Embedded Project* dialog, those definitions will be stored as part of the Embedded project. While the Auto-Import option is not enabled, the dialog will always use the project's stored definitions. When the Auto-Import option is enabled (and the Embedded and FPGA projects linked), these project-based definitions will be by-passed, in favor of those definitions stored as part of the processor component in the FPGA design.

Allocating Address Space in a 32-bit Processor

Sections and Reserved Areas

Once the view of the physical memory devices has been defined from the processor's perspective, it is then possible to further sub-divide these into logical sections. This allows the embedded software tools to place different parts of the software application:

- into different physical memory devices and
- into different locations within a single physical memory device.

These named "sections" can then be referenced from assembler and C source code, to enable code and data to be placed at certain memory locations. This allows for fine control over how the program is located or uses memory.

As well as defining areas in which to place code 'sections', you can also specify areas of memory that are off-limits to the Linker – reserved areas in which code cannot be placed (analogous to the use of keepouts from a physical board-level perspective).

Sections and reserved areas are defined and managed from the **Sections/Reserved Areas** tab of the *Options for Embedded Project* dialog (Figure 9). Access this dialog by right-clicking on the project's entry in the *Projects* panel and choosing **Project Options** from the context menu that appears. This tab can be also accessed directly from the *Component Properties* dialog for the processor, from within the FPGA design.

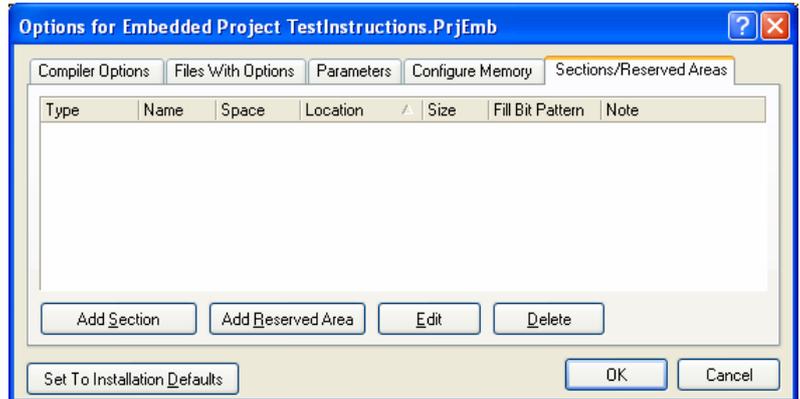


Figure 9. Defining Sections and Reserved Areas as part of Embedded Project options.

Controls are provided, both in terms of buttons and a right-click menu, to add new Section and/or Reserved Area definitions, or edit/delete existing ones.

Sections

Click the **Add Section** button within the **Sections/Reserved Areas** tab to access the *Section* dialog (Figure 10).

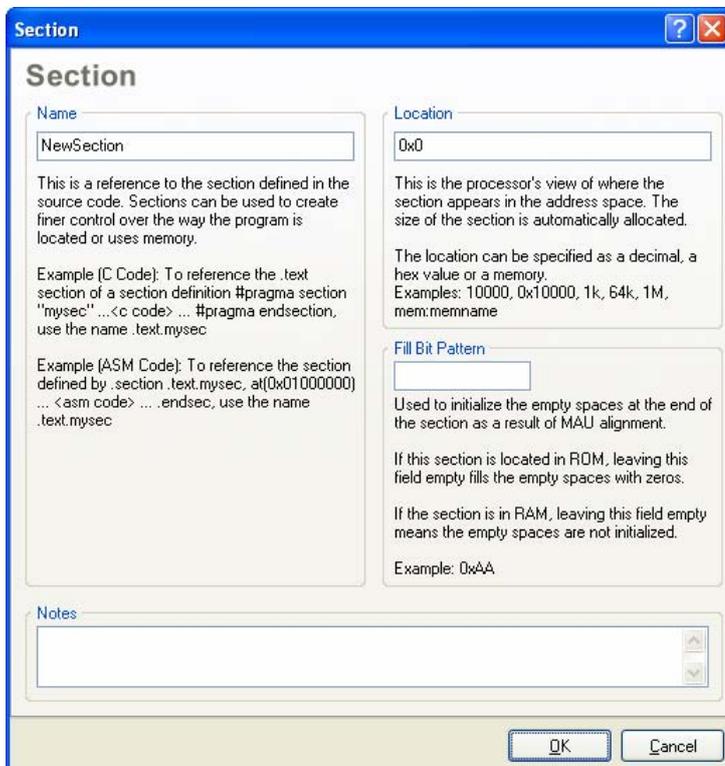


Figure 10. Defining a section within processor address space.

Use the **Name** region of the dialog to reference the required section defined in the source code. For example, if the source code is written in assembly code (* .asm) and the following section is defined:

```
.section.text.shiftcontrol, at(0x02000000)
.
.
Code Statements
.
.
.endsec
```

then the name specified in the *Section* dialog in order to reference this section definition would be `.text.shiftcontrol`.

The **Location** region allows you to specify where in the processor's address space the section is to be located. Simply enter a base address for the section – the size will be automatically allocated. You can specify the location using decimal or Hex notation (e.g. 10000, 0x10000, 1k, 64k, 1M). Alternatively, you can specify the location as a specific memory, in the format `mem:memname`.

The **Fill Bit Pattern** region allows you to specify a value to be used to initialize the empty spaces at the end of the section, as a result of MAU alignment. Leaving this field empty will:

- Fill the empty spaces with zeros if the section is located in ROM
- Leave the empty spaces uninitialized, if the section is located in RAM.

Use the **Notes** section to add any comments, such as the purpose for the section, when it was added, who it was added by, etc.

Reserved Areas

Click the **Add Reserved Area** button within the **Sections/Reserved Areas** tab to access the *Reserved Area* dialog (Figure 11).

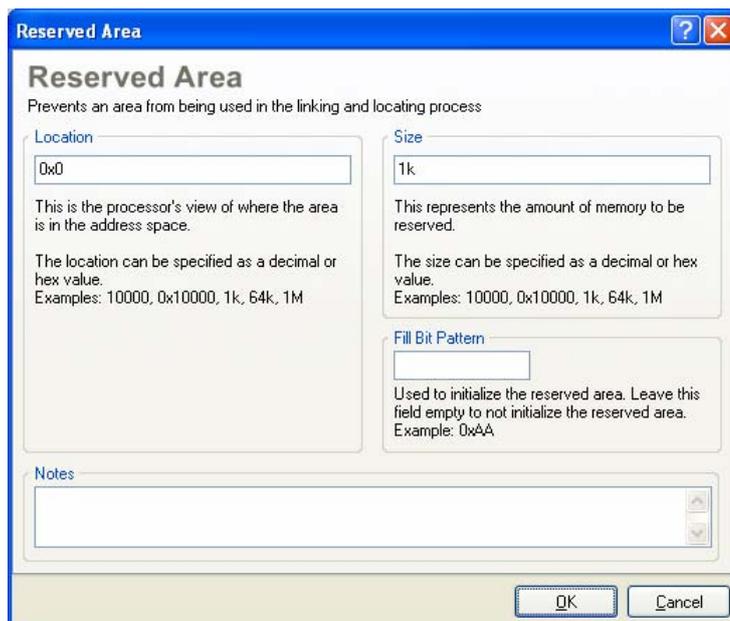


Figure 11. Defining a Reserved Area within processor address space.

The **Location** region allows you to specify where in the processor's address space the reserved area is to be located. Simply enter a base address for the area, using decimal or Hex notation (e.g. 10000, 0x10000, 1k, 64k, 1M).

The **Size** region of the dialog enables you to specify how much memory should be reserved using this area definition. Enter the value as required – again decimal or Hex notation can be used (e.g. 10000, 0x10000, 1k, 64k, 1M).

Use the **Fill Bit Pattern** region to specify a value with which to initialize the reserved area, if required.

Use the **Notes** section to add any comments, such as when the area was added, who it was added by, etc.

Configuring Processor Peripheral I/O

Typically, the required peripheral devices will be added to the design and wired to the processor's Peripheral I/O interface accordingly, before mapping is performed. To illustrate, consider the circuitry of Figure 12.

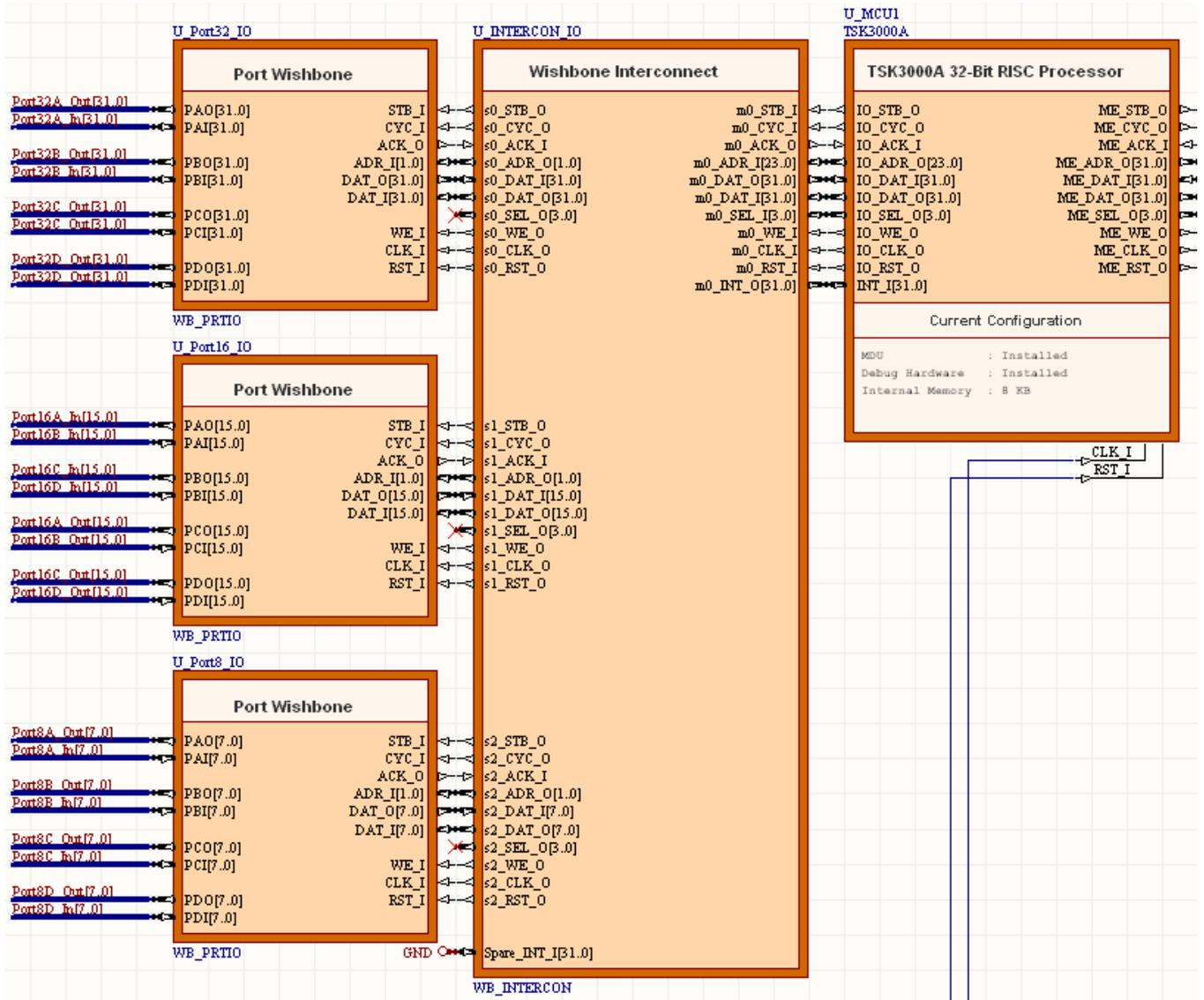


Figure 12. Connection of peripheral I/O devices to the 32-bit processor.

Here, three configurable Wishbone port devices (configured with 8-, 16- and 32-bit data widths respectively) have been connected to a TSK3000A 32-bit processor. Connection to the processor's External Peripheral I/O interface is made through a configurable Wishbone Interconnect device (WB_INTERCON).

For more information on connecting peripheral I/O devices to a processor, refer to the application note [Connecting Memory and Peripheral Devices to a 32-bit Processor](#).

Mapping of slave peripheral I/O devices into the processor's address space is performed only from within the FPGA project, using the *Configure Peripheral* dialog (Figure 13). Access this dialog by right-clicking on the symbol for the processor (in the schematic) and choosing **Configure Processor Peripheral** from the context menu that appears. Alternatively, use the **Configure Processor Peripheral** button available from the *Component Properties* dialog for the processor.

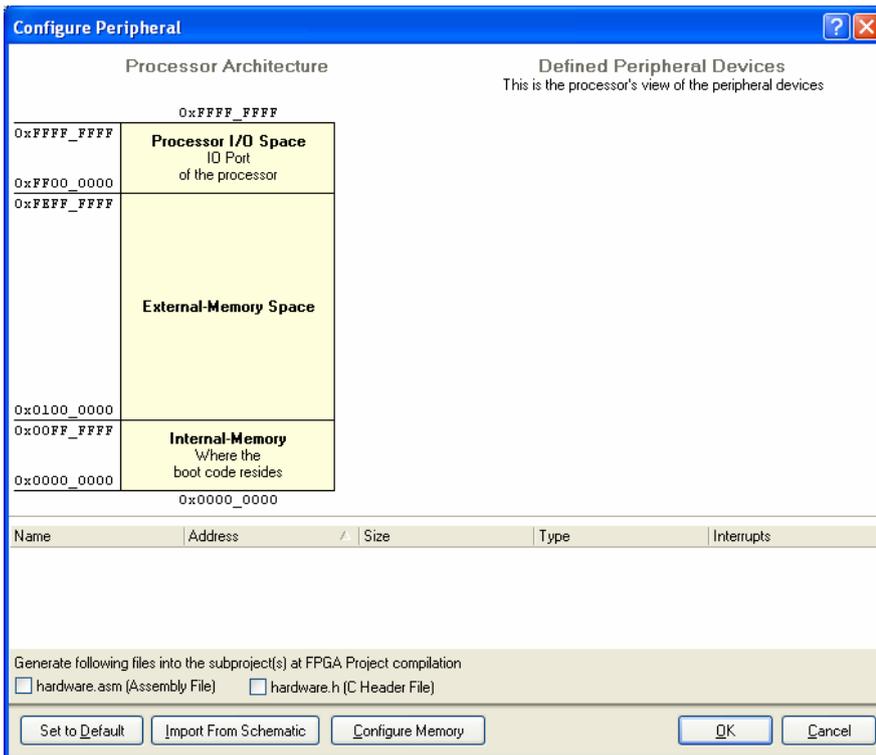


Figure 13. Mapping peripheral devices into processor memory address space

The dialog is divided into two regions. The top region provides a visual summary of the peripheral devices that are currently defined and mapped into the processor's address space – i.e. the processor's 'view' of the actual peripherals in the design. The bottom region of the dialog provides a tabular listing of each of these defined peripheral devices.

Controls are also provided in the dialog for generating header files (Assembly or C) upon compilation of the FPGA project.

The dialog also provides a **Configure Memory** button – allowing you to quickly jump between peripheral and memory configuration dialogs. Note that when using this facility, you will be prompted to confirm whether you wish to save the configuration of the current dialog before proceeding to the other dialog.

Importing Device Information from the FPGA Project Schematic

If you have connected your slave peripheral device(s) to the processor through a Wishbone Interconnect, then the simplest way to make these devices 'known' to the processor – and automatically have them mapped into the processor's address space – is to use the *Configure Peripheral* dialog's **Import from Schematic** feature.

Launched by clicking on the **Import From Schematic** button, the feature essentially scans the FPGA design for configurable Wishbone Interconnect devices. All detected interconnect devices, along with the slave devices associated to them, are presented in the *Choose Wishbone Items* dialog (Figure 14).

Prior to the dialog appearing, you will be given the choice of whether or not to delete any existing peripheral device definitions that have been added. If you have not yet configured the processor I/O space, or want to start over again from a 'blank canvas' as it were, then opting to delete here is the best course of action.

On the other hand, you may have already defined peripherals that you wish to keep. This can be the case when adding additional peripheral devices to an already well-established design. In this case, it is better to not to delete the existing definitions, but rather import just the addition(s).

Each slave peripheral device connected to an Interconnect is listed in the dialog, in terms of its identifier. This identifier is assigned as part of the slave device's definition when configuring the Interconnect. The dialog offers the ability to import selected devices or all devices associated to a given Interconnect, by clicking on the relevant entry in the **Import to Bus** column. Choose the peripheral devices you wish to import and click **OK**.

Importing will:

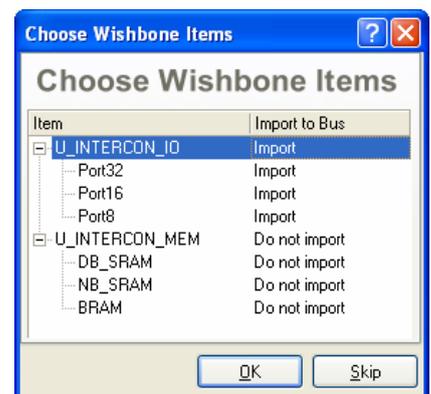


Figure 14. Selecting peripheral devices to import into processor address space.

Allocating Address Space in a 32-bit Processor

- Add peripheral device definitions where they currently do not exist. The identifier of an Interconnect slave device – defined when configuring the Interconnect – will be used to name the peripheral device definition.
- Update any existing peripheral device definitions, only if the identifier of the Interconnect slave device matches that of an existing peripheral device definition.
- Preserve existing peripheral device definitions whose identifiers do not match those of the Interconnect slave devices being imported – leaving hand-crafted definitions untouched.

Update to/preservation of existing peripheral definitions, will only be possible where you have opted to not delete the existing configuration before importing.

 Only peripheral devices connected to the processor via a Wishbone Interconnect device will be imported using the Import From Schematic feature. If a single peripheral device is connected directly to the processor's Peripheral I/O interface, it will not be imported and mapped. In this case, you will need to manually add a definition for the device.

Figure 15 illustrates an example of this mapping based on the example schematic circuitry of Figure 12.

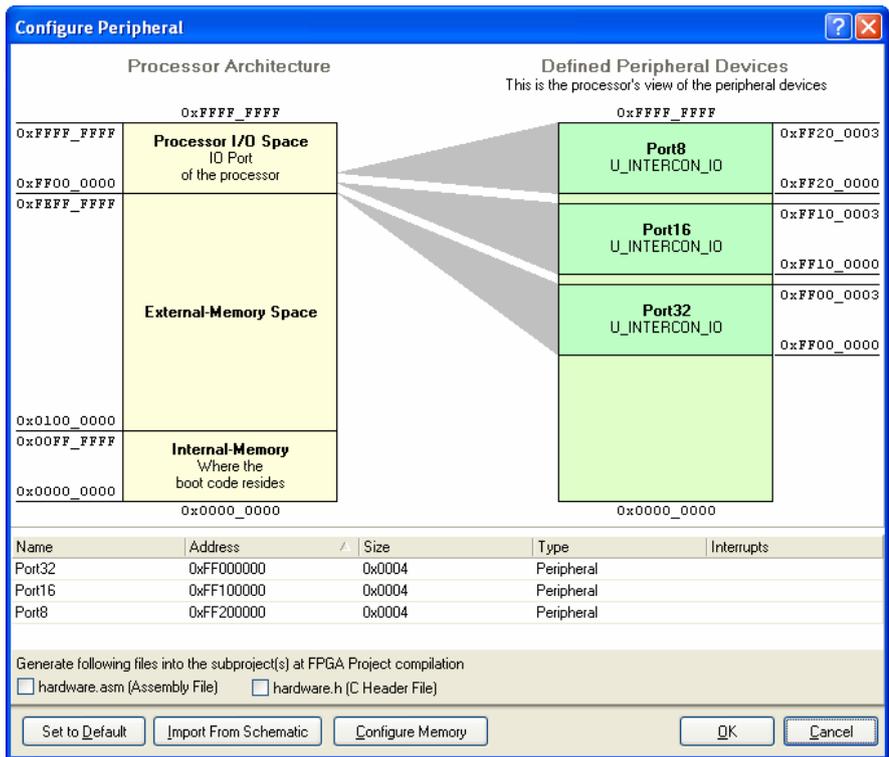


Figure 15. Defining peripheral devices automatically by importing from the schematic.

If you do not wish to import any Interconnect-related devices, simply click on the **Skip** button (in the *Choose Wishbone Items* dialog).

Managing Peripheral Device Definitions

New peripheral device definitions may be added, or existing ones modified or deleted, using the available commands on the right-click menu for the *Configure Peripheral* dialog. An existing peripheral device can also be modified with respect to its definition by double-clicking on its corresponding entry in the list section of the dialog, or by double-clicking on its 'block' in the graphical display. The associated *Peripheral Device* dialog will appear (Figure 16).

The dialog enables you to accurately define the peripheral device in terms of its size and base address – specifying where in the processor's address space it is to be mapped. You can specify these values using decimal or Hex notation (e.g. 10000, 0x10000, 1k, 64k, 1M).

The **Name** field allows you to define a unique identifier for the device. The identifier for each peripheral device will be used when generating header files for inclusion into the Embedded Software project (see next section). It is important to note that the identifier used must not contain spaces.

The Type region allows you to specify the type of peripheral. Choose between the following options:

- **Peripheral** – standard peripheral device.
- **Peripheral ASP** – a peripheral whose functionality is described in the Embedded Software project.

The **Interrupts** region of the dialog allows you to assign the required interrupt lines to the slave device. For devices that generate multiple interrupts, assignment is made by entering a comma-separated list.

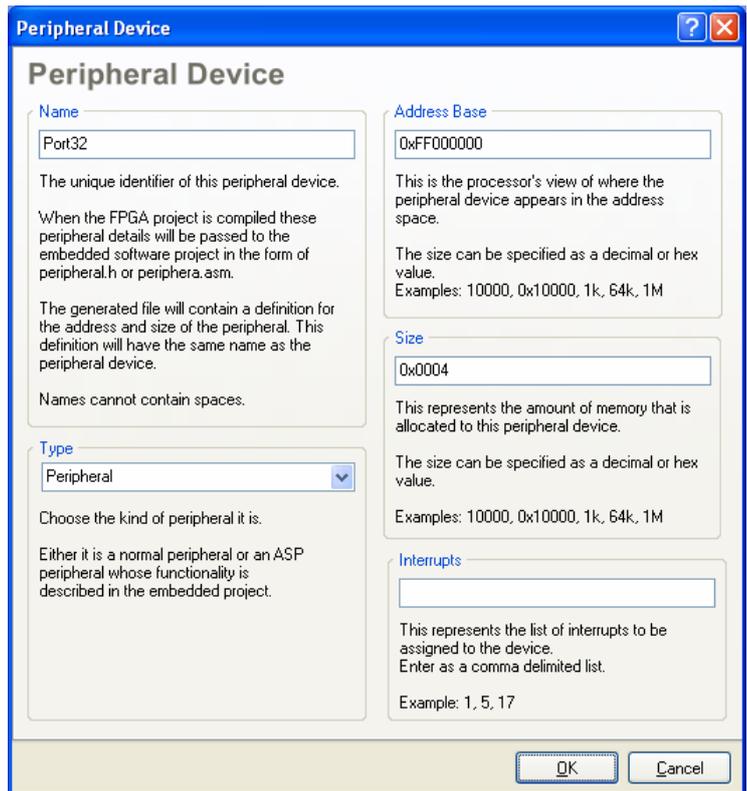


Figure 16. Defining a mapped peripheral I/O device.

Generating Header Files for the Embedded Software Project

When mapping slave memory and peripheral I/O devices to the processor's address space from within the FPGA project, the *Configure Processor Memory* and *Configure Peripheral* dialogs provide two options for passing the memory and peripheral definition information to the Embedded Software project:

- hardware.asm (Assembly File)
- hardware.h (C Header File)

The option you choose will depend on whether the embedded source code is written in Assembly or C. In each case, when the FPGA project is compiled, the chosen definition file will be created and added to the Embedded Project.

A header file essentially provides keyword substitution. The Embedded Software designer can simply enter a definition into the code, which, through the header file, will be substituted with the required information at compile time. It is far easier to remember and use identifiers for device addresses and sizes, rather than the hexadecimal representations themselves.

For each mapped memory and peripheral device, the file will contain the following definitions:

- The device's base address. The name for this entry will appear in the form *Base_DeviceIdentifier* (e.g. *Base_EMAC32*).
- The device's size. The name for this entry will appear in the form *Size_DeviceIdentifier* (e.g. *Size_EMAC32*).

For those peripheral devices that generate interrupts to the processor, the corresponding interrupt definitions will also be listed. The name for an interrupt entry will be of the form:

Intr_DeviceIdentifier_InterruptIndex

where *InterruptIndex* is an alpha suffix (A, B, C, etc) to distinguish between multiple interrupts from the same device. Consider for example a BT656 Video Capture Controller, with the identifier *Video*, and which generates two interrupts to the host 32-bit processor. The interrupt name entries for this device in a generated header file would be:

```
Intr_Video_A
Intr_Video_B
```

Figure 17 shows an example of two generated header files – one C, the other Assembly.

Enabling or disabling an option in one dialog will automatically enable/disable the corresponding option in the other dialog.

DeviceIdentifier is the unique name assigned to the memory or peripheral device as part of its definition when mapping.

Allocating Address Space in a 32-bit Processor

```

hardware.h
//.....
// Automatically generated header file.
// Generated: 10:47:45 PM 11/07/2006
// This file should not be edited.
//.....

#ifndef __HARDWARE_H__
#define __HARDWARE_H__

//.....
#define Base_LEDS          0xFF000000
#define Size_LEDS         0x00000001
//.....

//.....
#define Base_SRL0         0xFF300000
#define Size_SRL0        0x00000010
#define Intr_SRL0_A      2
//.....

//.....
#define Base_EMAC32      0xFF100000
#define Size_EMAC32     0x00000020
#define Intr_EMAC32_A    0
//.....

//.....
#define INTERRUPT_CONTROL_CFG  0x00000005
#define INTERRUPT_KINDS_CFG    0x00000000
#define INTERRUPT_EDGE_KIND_CFG 0x00000000
#define INTERRUPT_LVL_KIND_CFG  0x00000005
//.....

//.....
#define Base_U_MCU1        0x00000000
#define Size_U_MCU1       0x00008000
//.....

//.....
#define Base_EXT_RAM      0x01000000
#define Size_EXT_RAM     0x00100000
//.....

#endif // __HARDWARE_H__

hardware.asm
;.....
; Automatically generated header file.
; Generated: 10:47:45 PM 11/07/2006
; This file should not be edited.
;.....

;.....
Base_LEDS          .equ 0xFF000000
Size_LEDS         .equ 0x00000001
;.....

;.....
Base_SRL0         .equ 0xFF300000
Size_SRL0        .equ 0x00000010
Intr_SRL0_A      .equ 2
;.....

;.....
Base_EMAC32      .equ 0xFF100000
Size_EMAC32     .equ 0x00000020
Intr_EMAC32_A    .equ 0
;.....

//.....
#define INTERRUPT_CONTROL_CFG  0x00000005
#define INTERRUPT_KINDS_CFG    0x00000000
#define INTERRUPT_EDGE_KIND_CFG 0x00000000
#define INTERRUPT_LVL_KIND_CFG  0x00000005
//.....

;.....
Base_U_MCU1        .equ 0x00000000
Size_U_MCU1       .equ 0x00008000
;.....

;.....
Base_EXT_RAM      .equ 0x01000000
Size_EXT_RAM     .equ 0x00100000
;.....

```

Figure 17. Example of generated hardware definition files for use by the embedded software.

Revision History

Date	Version No.	Revision
14-Jul-2006	1.0	Initial release
27-Feb-2008	2.0	Updated for Altium Designer Summer 08

Software, hardware, documentation and related materials:

Copyright © 2008 Altium Limited.

All rights reserved. You are permitted to print this document provided that (1) the use of such is for personal use only and will not be copied or posted on any network computer or broadcast in any media, and (2) no modifications of the document is made. Unauthorized duplication, in whole or part, of this document by any means, mechanical or electronic, including translation into another language, except for brief excerpts in published reviews, is prohibited without the express written permission of Altium Limited. Unauthorized duplication of this work may also be prohibited by local statute. Violators may be subject to both criminal and civil penalties, including fines and/or imprisonment. Altium, Altium Designer, Board Insight, Design Explorer, DXP, LiveDesign, NanoBoard, NanoTalk, P-CAD, SimCode, Situs, TASKING, and Topological Autorouting and their respective logos are trademarks or registered trademarks of Altium Limited or its subsidiaries. All other registered or unregistered trademarks referenced herein are the property of their respective owners and no trademark rights to the same are claimed.