

WB_ASP - Configuration

Modified by on 6-Nov-2013

The WB_ASP can be configured after placement on the OpenBus System document, or schematic sheet, using the associated *Configure (WB_ASP Properties)* dialog (Figure 1). Access to this dialog depends on the document in which you are working:

- **In the OpenBus System document** – access the dialog by right-clicking over the component and choosing the Configure command from the menu that appears. Alternatively, double-click on the component to access the dialog directly.
- **In the Schematic document** – simply right-click over the device and choose the Configure command from the context menu that appears. Alternatively, click on the Configure button, available in the *Component Properties* dialog for the device.

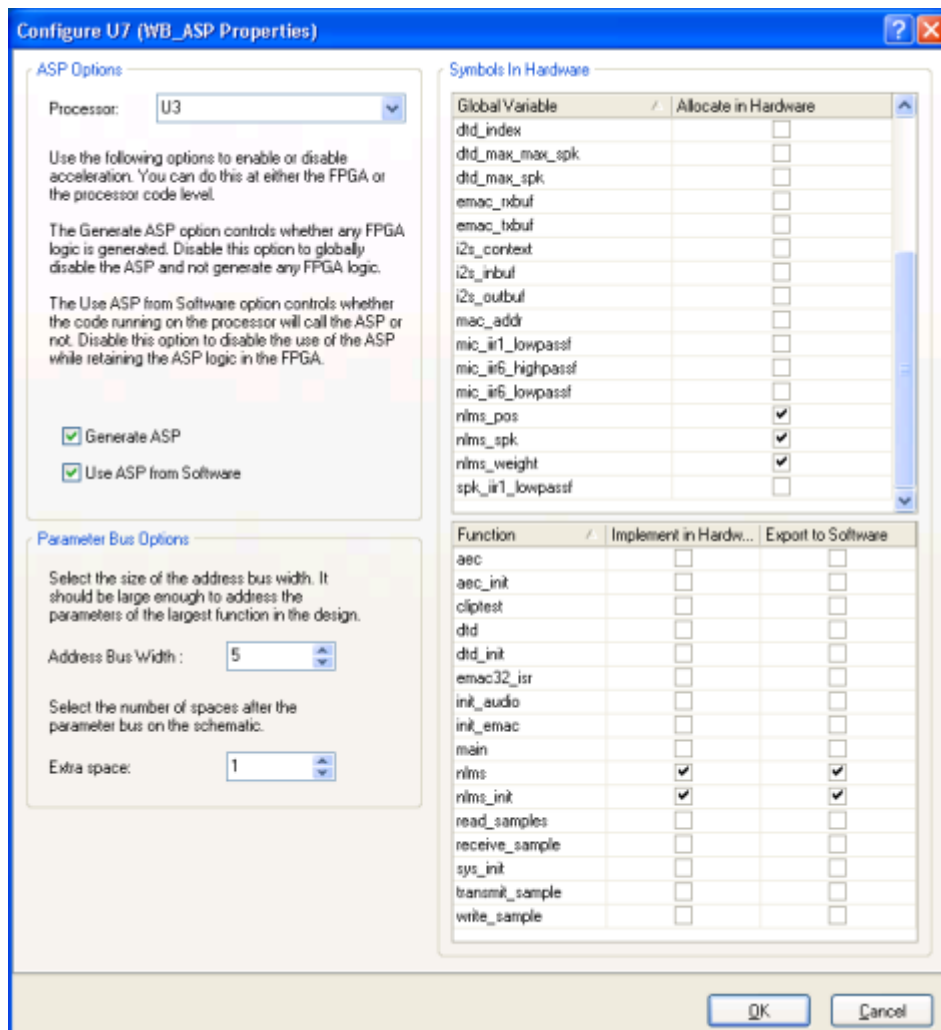


Figure 1. Configuring the ASP peripheral.

Use the dialog to configure the WB_ASP peripheral in accordance with design requirements. The dialog is divided into three areas. The following sections take a closer look at the options available in each of these areas.

ASP Options

The **Processor** field in this area of the dialog allows you to specify the processor that is connected to the WB_ASP and which can therefore call functions that have been generated in hardware. If your design contains multiple processors, simply choose the required processor (by designator) from the drop-down field.

The **Generate ASP** option provides the ability to enable or disable generation of hardware-compiled functions. With this option enabled, the C-to-Hardware Compiler will be invoked when you compile and synthesize your design project. All functions that have been enabled for implementation in hardware will be created as electronic circuits in the FPGA fabric.

The **Use ASP from Software** option enables you to control, on a global level, whether functions compiled into hardware will be called by software-based functions running within the processor. If this option is disabled, the embedded compiler will generate the functions in software and these will be used.

In terms of code development, debugging a function is only really possible at the C source code level – it is near-impossible to debug the corresponding hardware implementation of that function. By enabling the **Generate ASP** option, and disabling the **Use ASP from Software** option, you can effectively test and develop the software-compiled code only. The FPGA logic for any functions enabled for implementation in hardware will still be generated, but the processor will only use software-compiled versions of those functions. Once the software is fully debugged and operates as required, simply enable the **Use ASP from Software** option to switch over to using the hardware implementations of those functions.

The VHDL or Verilog hardware implementation of a function, generated by the C-to-Hardware Compiler, is "correct by construction". As long as the C code is fully debugged and error free, then the generated HDL code will also be free of errors.

The **Use ASP from Software** option can also be enabled/disabled at the source code level, using the **C To Hardware** panel (Figure 2).

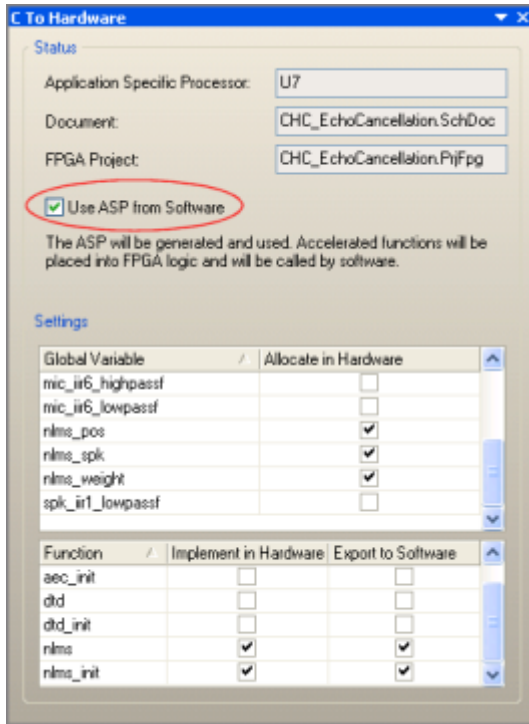


Figure 2. Globally controlling use of hardware functions at the code level.

If a design has been processed with the **Generate ASP** option enabled, then if the state of the **Use ASP from Software** option is changed, you only need to recompile and download the updated embedded software. Full reprocessing of the entire FPGA project is not required as the logic for the hardware functions already exists. In this way you can quickly switch between software-only and software-hardware implementations of the design, to observe the benefits obtained by using hardware acceleration.

Parameter Bus Options

This area of the dialog allows you to define the width of the WB_ASP's host interface address bus (io_ADR_I). The address bus is used by the processor to access and pass values to parameters in a hardware function, and also to access and read back a function's return value, where applicable.

The CHC Compiler will actually generate the minimum number of address lines required, but in order to wire the WB_ASP device correctly into the FPGA design, the width of the bus must be defined 'ahead of time'.

Use the **Address Bus Width** field to specify the width required. The width should be large enough to include all parameters of the largest function being implemented in hardware (and accessed from software). For example, consider the following functions destined to be compiled into hardware:

```
void chc_plot_init(int16_t xx, int16_t xy, int16_t xz, int16_t yx, int16_t yy,
int16_t yz, int16_t zx, int16_t zy, int16_t zz, uint16_t* vgabuf, int16_t*
zbuf);
```

```
void chc_plot_cube(uint16_t* p1, uint16_t* p2, uint16_t* p3, uint16_t* p4,
uint16_t* p5, uint16_t* p6);
```

```
void chc_zbuf_clear(int16_t* zbuf);
```

```
void chc_screen_clear(uint16_t* zbuf);
```

The largest of these functions, in terms of associated parameters, is `chc_plot_init`. The total parameters for this function is 11. In terms of addressing, it is not readily-apparent how wide the bus must be, as the size of the address space depends on the number of parameters involved and the data type of those parameters. The approach when designing is to start with the default address bus size of 4 bits. This will be more than enough for most parameters in typical hardware-implemented functions. In a schematic-based design, this setting will be reflected in the host interface address pin - `io_ADR_I`.

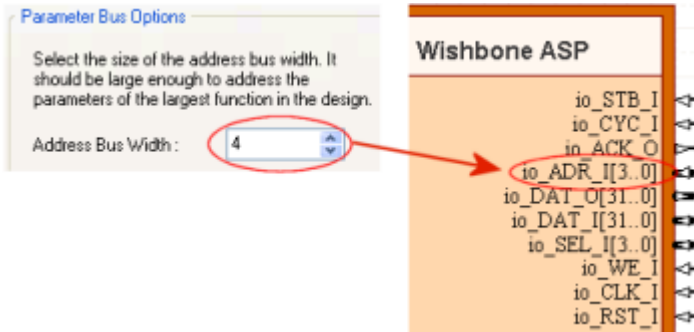


Figure 3. Defining address bus width for the host interface.

Upon compilation, if the required address bus width is greater than that specified, the C-to-Hardware Compiler will flag an error. In this case, simply reconfigure the WB_ASP with an increased value for the address bus width.

If your FPGA design is schematic-based, this area of the dialog also enables you to specify the amount of blank space between the host processor and external memory interfaces of the device. This can be used to great effect to simplify wiring and enhance readability. Simply use the **Extra space** field as required to increase or decrease the space.

Symbols In Hardware

This area of the dialog is divided into two lists. The upper list reflects all global variables present in the linked embedded software project.

Global Variable	Allocate in Hardware
mic_ir6_highpassf	<input type="checkbox"/>
mic_ir6_lowpassf	<input type="checkbox"/>
nlms_pos	<input checked="" type="checkbox"/>
nlms_spk	<input checked="" type="checkbox"/>
nlms_weight	<input checked="" type="checkbox"/>
spk_ir1_lowpassf	<input type="checkbox"/>

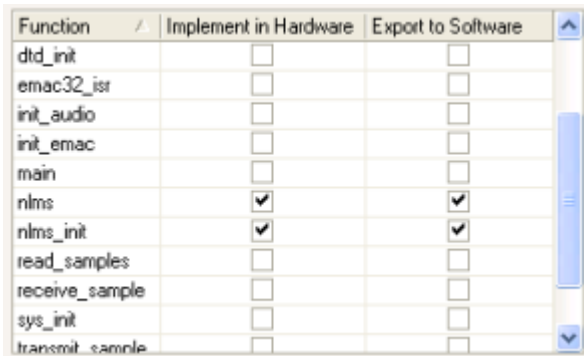
Figure 4. Allocating global variables in hardware.

If you want to have a variable allocated in hardware, simply enable the corresponding check box in the **Allocate in Hardware** column. Such a variable will be allocated in ASP block RAM by the CHC Compiler. Access to this memory is much faster, in comparison to storage allocation in block RAM outside of the ASP by the Embedded Compiler.

The list supports standard multi-select features (**Shift**+click, **Ctrl**+click and click & drag). This enables you to quickly select multiple variables. Once selected, use the available right-click context menu commands to quickly enable (**Push to Hardware**) or disable (**Remove from Hardware**) the corresponding **Allocate in Hardware** option.

A global variable that is allocated in hardware can only be accessed by a function that has also been implemented in hardware. Such a variable can not be called from a software-based function running on the host processor.

The lower list in this region of the dialog reflects all functions present in the linked embedded software project.



Function	Implement in Hardware	Export to Software
dtd_init	<input type="checkbox"/>	<input type="checkbox"/>
emac32_isr	<input type="checkbox"/>	<input type="checkbox"/>
init_audio	<input type="checkbox"/>	<input type="checkbox"/>
init_emac	<input type="checkbox"/>	<input type="checkbox"/>
main	<input type="checkbox"/>	<input type="checkbox"/>
nims	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
nims_init	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
read_samples	<input type="checkbox"/>	<input type="checkbox"/>
receive_sample	<input type="checkbox"/>	<input type="checkbox"/>
sys_init	<input type="checkbox"/>	<input type="checkbox"/>
transmit_sample	<input type="checkbox"/>	<input type="checkbox"/>

Figure 5. Specifying which functions are to be implemented in hardware.

If you want to implement a function in hardware – generated by the CHC Compiler as part of the ASP – simply enable the corresponding check box in the **Implement in Hardware** column. Should you wish to be able to call that hardware function from within the software running on the host processor, ensure that the corresponding check box in the **Export to Software** column is also enabled.

To summarize the effect of using these two options:

- A function with enabled **Implement in Hardware** option will become a hardware function
- A hardware function can call another hardware function
- A hardware function can not call a software function
- A software function running on the host processor can call a hardware function, provided that hardware function has been exported to software (**Export to Software** option enabled for function)
- A hardware-only function (not exported to software) can call a hardware function that has been exported to software. Like-wise, a hardware function that has been exported to software can call a hardware-only function.

The list supports standard multi-select features (**Shift**+click, **Ctrl**+click and click & drag). This enables you to quickly select multiple functions. Once selected, use the available right-click context menu commands to:

- **Push to Hardware** – enable Implement in Hardware option for each function in the selection.
- **Remove from Hardware** – disable Implement in Hardware option for each function in the selection.
- **Push and Export to Hardware** – enable Implement in Hardware and Export to Software options for each function in the selection.

- **Unexport from Hardware** – disable Export to Software option for each function in the selection.

Allocation of variables and implementation of functions in hardware can also be performed from within the C source code – either from the **C To Hardware** panel, or by right-clicking on a global variable/function directly in the code editor and using the relevant commands that appear on the context menu. When using the panel, only global variables and functions defined in the active C document will be listed.

Source URL: http://techdocs.altium.com/display/FPGA/WB_ASP+-+Configuration#comment-0