Overview of DesignWare Synthesizable IP Components for AMBA 3 AXI

AMBA 3.0 (AXI) is a standard bus architecture system developed by ARM designed for high-performance, high-frequency system designs. The DesignWare solution for AMBA 3 AXI provides a portfolio of synthesizable IP to aid designers in the rapid and accurate development of AXI-based components and system-on-chip (SoC) designs. Each component is packaged using Synopsys' coreTools technology to reduce integration risks and provide flexible, reliable configuration. Using AXI synthesizable IP from Synopsys DesignWare enables designers to focus on the high-value portion of their product, while significantly reducing time to market and development costs.

Existing Components

The following components have been released:

- DW_axi (page 5) – This is a multiple address, multiple data interconnect, where simultaneous transfers between different AXI masters and AXI slaves can occur.
- DW_axi_gm (page 10)– This is an interface module that provides a method to transfer transactions from a master with a generic interface (GIF) to the AXI bus. DW_axi_gm simplifies the connection of custom or third-party master controllers to the AXI bus.
- DW_axi_gs (page 9) – This is an interface module that provides a method to transfer transactions from the AXI bus to a more basic GIF slave. DW_axi_gs simplifies the connection of custom or third-party slave components to the AXI bus.
- DW_axi_x2h (page 11) – This is an AXI-to-AHB bridge, which maps the AMBA AXI protocol to the AMBA AHB protocol. On one side, the DW_axi_x2h acts as an AXI Slave and is connected to an AXI interconnect or directly to an AXI Master. On the other side, the DW_axi_x2h acts as an AHB Master and is connected to an AHB interconnect.
- DW_axi_rs (page 12) – This component pipelines AXI channels to terminate critical timing paths.

- DW_axi_hmx (page 12) – This is an interface module that connects an AHB master to an AXI subsystem.
- DW_axi_x2p (page 13) – This bridge connects an AXI subsystem to an APB 3.0 subsystem.
- DW_axi_x2x (page 14) – This bridge connects an AXI master and an AXI slave.

New Components

There are no AXI components currently under development.

Application Notes

In addition to the new component releases,the following application notes have been written:

- "Connecting an AMBA 2.0 AHB Slave to an AMBA 3 AXI Subsystem" on page 17
- "Creating an AXI-to-APB 2.0 Bridge Using DesignWare IP for AMBA" on page 24
- "Connecting an AMBA 2.0 AHB Subsystem to an AMBA 3 AXI Subsystem" on page 30
- "Using DW_ahb_dmac in an AXI Subsystem" on page 37

Definitions

AXI subsystem – An AXI subsystem is implemented with an interconnect (DW_axi), which allows simultaneous transfers between different masters (up to 16) and slaves (up to 16). Master and slaves in an AXI subsystem are all in the same clock domain and have address and data channels of the same width.

Interconnect tiling – This allows the distribution of larger systems' interconnect with the purpose of increasing the system clock rates and distributing the physical interconnect. Note: When interconnects are tiled in this fashion, all interconnects belong to the same AXI subsystem—there is no bridging between the interconnects.

Bridges: A bridge connects subsystems together of same or different protocols, supports crossing of clock domains, and provides buffering. Some of the following bridges are existing or under development.

- **AXI-to-AHB (DW_axi_x2h)** – This bridge connects an AXI subsystem to an AHB subsystem.
- **AXI-to-AXI** (**DW_axi_x2x**) –This bridge not only connects different AXI subsystems together, but it can be used to connect an AXI master and AXI slaves of different port widths and/or different clock domains.
- **AXI-to-APB 3.0** (**DW_axi_x2p**) – This bridge connects an AXI subsystem to an AMBA APB 3.0 subsystem.
- **AXI-to-APB 2.0** – Currently, there are no plans to develop this bridge, but there is a way to obtain this functionality by using the DW_axi_x2h bridge and the DW_apb (AMBA 2.0 compliant) components. For more information about how to accomplish this, refer to "Connecting an AMBA 2.0 AHB Subsystem to an AMBA 3 AXI Subsystem" on page 30.

- **AHB-to-AXI** – Currently, there are no plans to develop this bridge, but there is a way to obtain this functionality by using the DW_axi_hmx and the DW_ahb_eh2h components. For more information about how to accomplish this, refer to"Connecting an AMBA 2.0 AHB Subsystem to an AMBA 3 AXI Subsystem" on page 30.

Register slice (DW_axi_rs) – This is a register used to pipeline AXI channels, allowing faster timing paths and higher system clock frequency.

Interconnect architecture – There are three AXI interconnect architectures: SASD (Single Address Single Data), SAMD (Single Address Multiple Data), and MAMD (Multiple Address Multiple Data).

- **Single address –** All slaves share the same read and write address bus - Masters most arbitrate for ownership when multiple masters target the same or different slaves in the same cycle.
- **Single data** – All slaves share the same write data bus - Masters most arbitrate for ownership when multiple masters target the same or different slaves in the same cycle (with write data). Also, all masters share the same read data and write response bus.
- **Multiple address –** All slaves have their own read and write address bus allowing parallel commands to different slaves in the same cycle.
- **Multiple data** – All slaves have their own write data bus - Also, all masters have their own read data and write response bus.

Visibility – Not all slaves need to be visible to all masters. For each master in a subsystem, connected together via an AXI interconnect (DW_axi), it is possible to configure whether that master has visibility to each slave in the subsystem—on a slave-by-slave basis. If a master does not have visibility to a slave, then the DW_axi interconnect does not provide any hardware to allow routing of traffic from that master to that slave.

AXI Layer – The AXI system can be viewed as a collection of visibility layers. On each layer there is an AXI master and all the slaves to which that master has visibility.

1 Existing AXI Components

The following DesignWare AXI components have been released. For more information about obtaining these components, contact Synopsys.

DW_axi

Multiple Address, Multiple Data AXI Interconnect

- Hybrid bus architecture
  - Configurable
  - MAMD (multiple address, multiple data) architecture
  - SASD (single address, single data) architecture
  - Mixture of both MAMD and SASD can be selected
  - Allows MAMD performance to be used where required, while low performance master slave links can be combined into shared layer to reduce area, power and wire routing
- Configurable number of master and slave ports (up to 16 each)
- Configurable system decoder (optional)
- Configurable master and slave priorities used for arbitration
- Default slave included
- Configurable slave visibility per master
- Built-in out-of-order deadlock avoidance
- Address width of 32 or 64 bits
- Data widths of 8, 16, 32, 64, 128, and 256 bits
- Equal data widths of master and slave ports
- Single clock frequency for all master and slave ports
- No buffering of AXI channel payload in interconnect
- Locked transfer support
- Automatically optimized in single AXI master subsystems
- Optional user sideband signals for each AXI channel, allowing additional control/status per channel (for example, data parity)

Figure 1 and Figure 2 on page 7 show a single DW_axi interconnect and tiling interconnects, respectively.

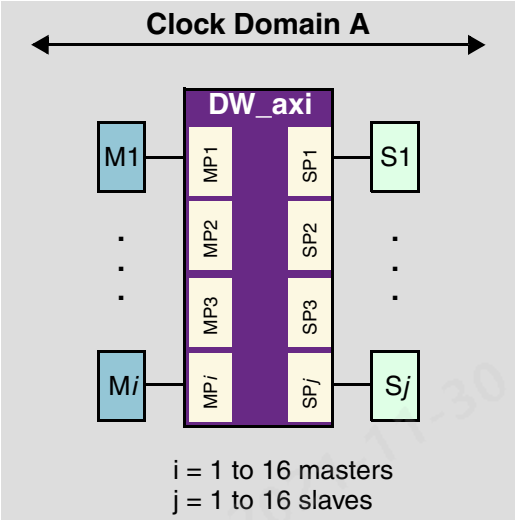


Figure 1: DW_axi Interconnect

By a combination of configuration and additional intermediate master/slave connections, multiple "tiled" interconnects can function the same as a single interconnect. This is illustrated in Figure 2 on page 7 and described in the "Example of Tiling Interconnects" on page 8. Tiling interconnects can result in higher system clock frequency and improved routing by distributing the physical interconnect wiring.

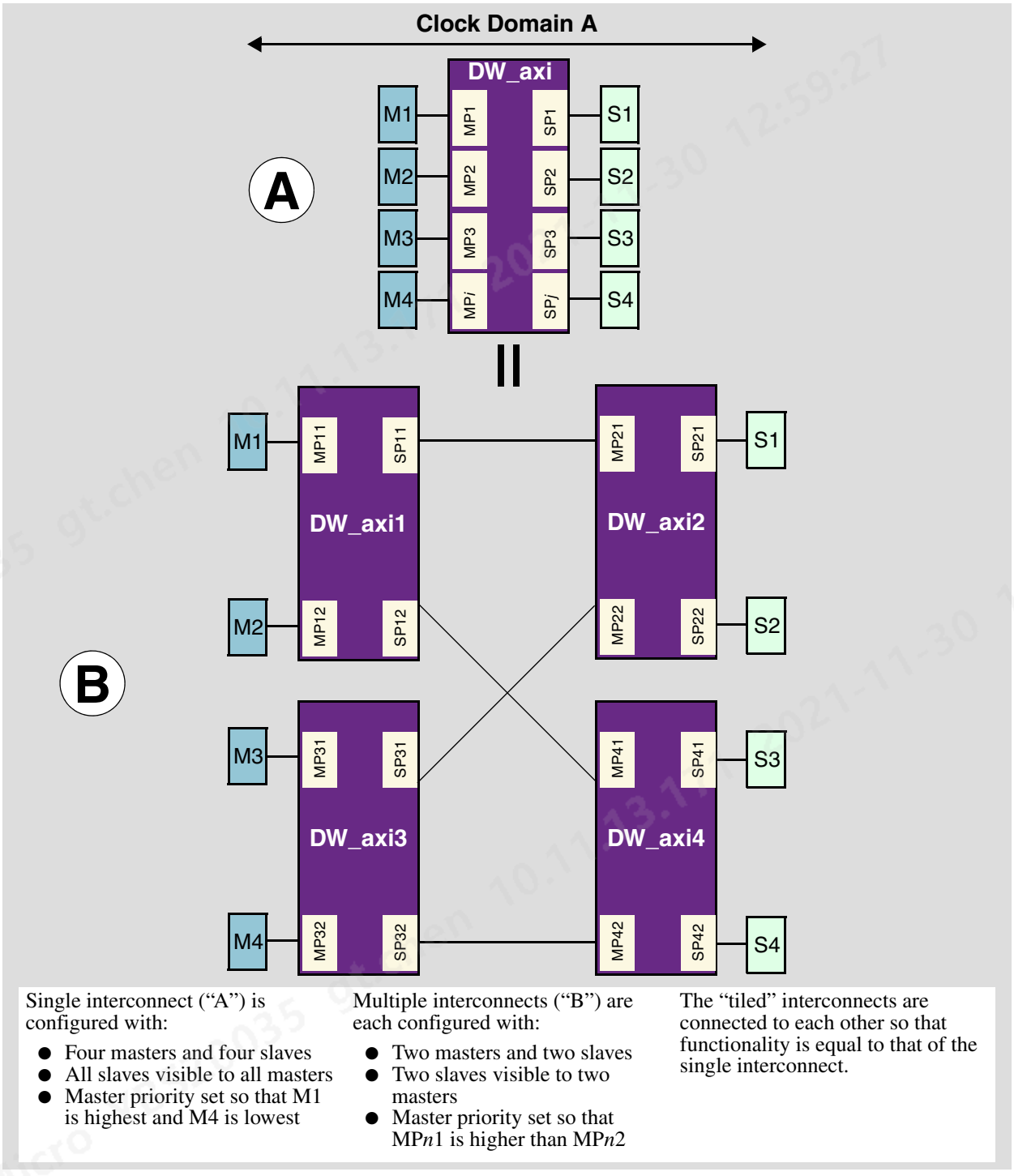


Figure 2: Tiling DW_axi Interconnects

Example of Tiling Interconnects

The four DW_axi interconnects illustrated in Figure 2 on page 7 are configured by setting the priority parameters in each of the DW_axi interconnects and connecting to the other DW_axi interconnects. In the example configuration/connectivity described below, the master priorities of the systems shown in Figure 1 on page 6 and Figure 2 on page 7 are made to be functionally similar. Other configuration/connectivity options are possible and left to the IP integrator to evaluate.

1. By configuration of priority parameters in each DW_axi:
   - DW_axi1 set MP11 higher priority than MP12
   - DW_axi2 set MP21 higher priority than MP22
   - DW_axi3 set MP31 higher priority than MP32
   - DW_axi4 set MP41 higher priority than MP42
2. By connecting the intermediate master/slave connections between the DW_axi instances:
   a. SP11 of DW_axi1 to MP21 of DW_axi2 – This gives M1 and M2 access to S1 and S2 through MP21, the highest priority port in DW_axi2. DW_axi1 has M1 at a higher priority than M2.
   b. SP12 of DW_axi1 to MP41 of DW_axi4 – This gives M1 and M2 access to S3 and S4 through MP41, the highest priority port in DW_axi4. At this point M1 and M2 have access to S1, S2, S3, S4, where M1 is a higher priority.
   c. SP31 of DW_axi3 to MP22 of DW_axi2 – This gives M3 and M4 access to S1 and S2 through MP22, a lower priority than MP21. M3 and M4 access S1 and S2 at a lower priority than M1 and M2.
   d. SP32 of DW_axi3 to MP42 of DW_axi4 – This gives M3 and M4 access to S3 and S4. M3 and M4 access S3 and S4 at a lower priority than M1 and M2.

So if:

- M1 and M2 have access to S1 and S2 through DW_axi2 port MP21 at the highest priority in DW_axi2;
- M1 and M2 have access to S3 and S4 through DW_axi4 port MP41 at the highest priority in DW_axi4;
- M3 and M4 have access to S1 and S2 through DW_axi2 port MP22 at a lower priority than M1 and M2 coming through MP21;
- M3 and M4 have access to S3 and S4 through DW_axi4 port MP42 at a lower priority than M1 and M2 coming through MP41; and
- M3 has a higher priority than M4; and
- M1 higher than M2

Then:

- All slaves are visible to all masters and the priority is M1, M2, M3, M4.

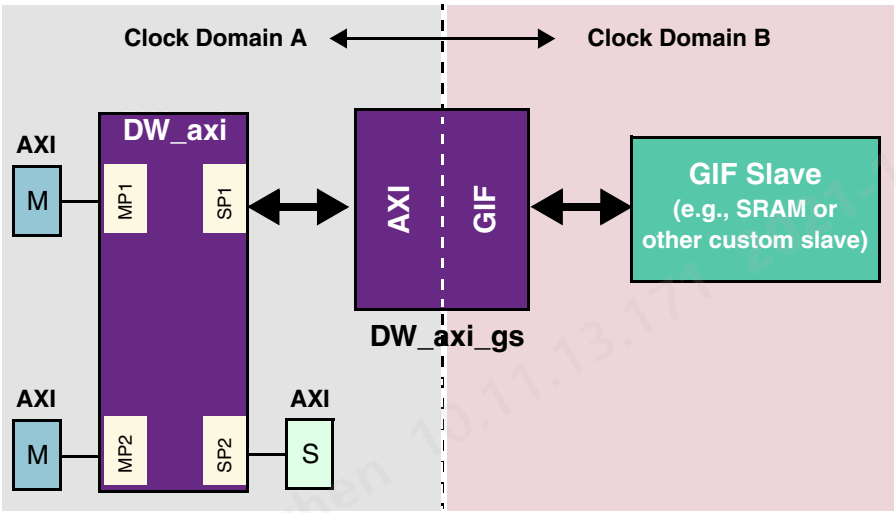
DW_axi_gs

AMBA AXI Slave to Generic Interface (GIF)

This module simplifies the connection of a custom or third-party slave component to the AMBA AXI bus using a Generic Interface (GIF). The GIF consists of two independent channels for requests and responses. The channels offer two-way flow control similar to the valid/ready handshake on the AXI bus.

As the DW_axi_gs accepts new transactions from the AXI, it transfers them to the GIF Request Channel and can be configured to buffer AXI transactions if the GIF is unable to immediately accept a transaction. Similarly, read data and write responses from the GIF Response Channel can be configured to be buffered if the AXI is unable to immediately accept a response. For very simple, fixed-latency slaves such as SRAM, a GIF-Lite configuration is available.

- Full support of the AXI protocol
- Configurable number of exclusive access monitors
- Full support of AXI low-power interface
- Two-way flow control
- Synchronous point-to-point communication on GIF
- Configurable number of outstanding transactions
- Synchronous clock support; including slower GIF clock
- Configurable microarchitecture
- Configurable I/O signals to support simple peripherals (such as, SRAM)



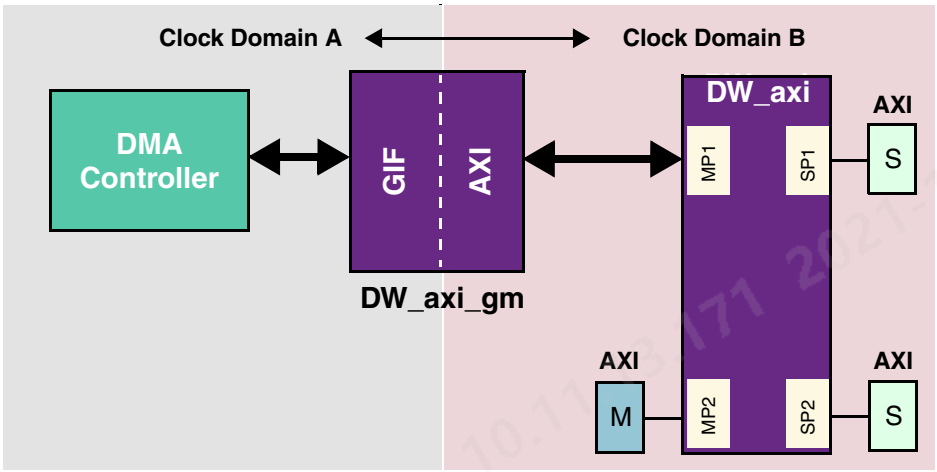
DW_axi_gm

GIF to AMBA AXI Master

This module simplifies the connection of a custom or third-party master controller to the AMBA AXI bus using a Generic Interface (GIF). The GIF consists of two independent channels for requests and responses. The channels offer two-way flow control similar to the valid/ready handshake on the AXI bus.

As the DW_axi_gm accepts new transactions from the GIF, it transfers them to the AXI master read or write channel and can be configured to buffer GIF transactions if the AXI is unable to immediately accept a transaction. Similarly, read data and write responses from the AXI interface can be configured to be buffered if the GIF is unable to immediately accept a response.

- Full support of the AXI protocol, except for exclusive accesses
- Full support of AXI low-power interface
- Two-way flow control
- Synchronous point-to-point communication on GIF
- Independent request and response GIF channels
- Unlimited outstanding transactions
- Synchronous clock support; including slower GIF clock
- Configurable micro-architecture
- Transaction block control

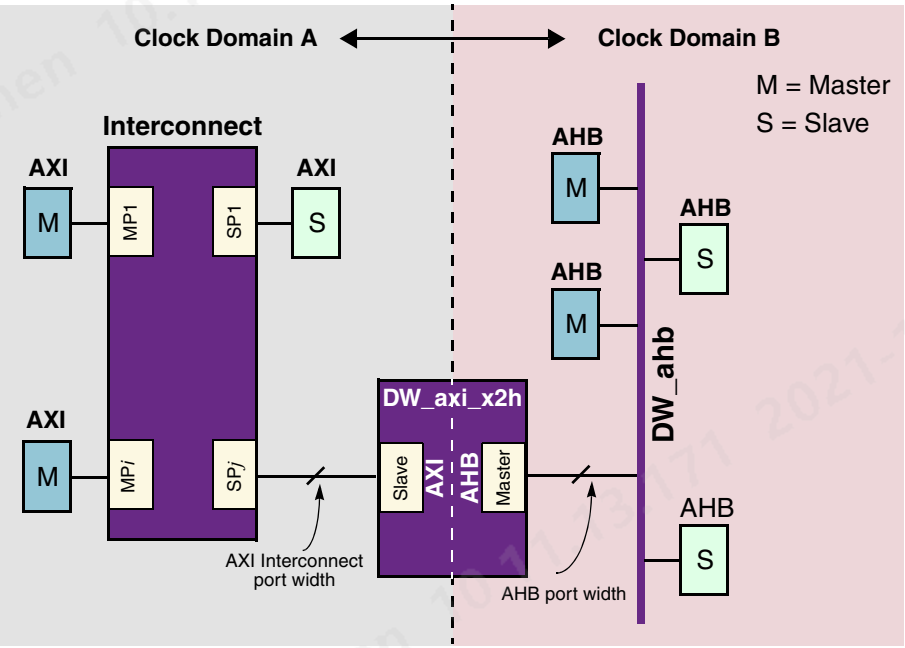


DW_axi_x2h

AXI-to-AHB Bridge

This component is used to connect an AXI subsystem to an AHB subsystem. DW_axi_x2h maps the AXI protocol to the AHB protocol. On one side, DW_axi_x2h acts as an AXI slave and is connected to an AXI interconnect or directly to an AXI master. On the other side, DW_axi_x2h acts as an AHB master and is connected to an AHB interconnect. An interface between the two sides is comprised of configurable buffers.

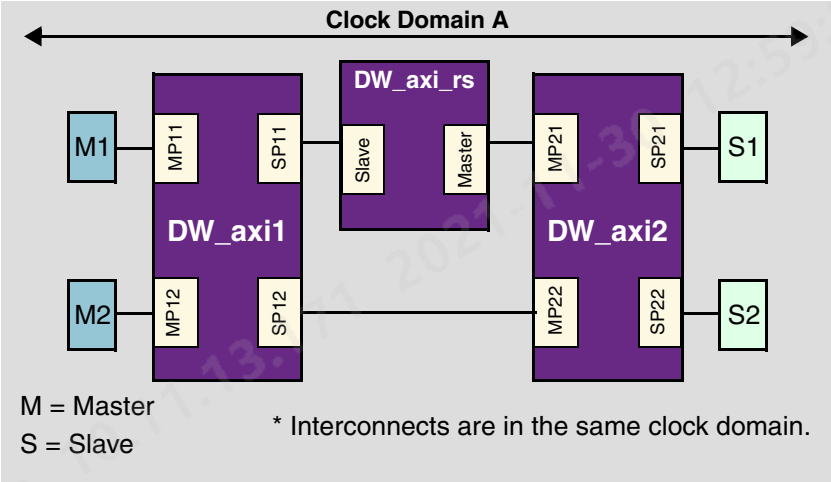
- Bridge from AXI to AHB bus, allowing for easy integration of legacy AHB designs with newer AXI designs
- Configurable AXI Slave interface
- Configurable AHB Master interface; includes AHB Lite support
- AMBA AHB and AXI compliant
- Configurable depths on command, data, and response queues
- All transactions passed through industry-standard DesignWare FIFOs
- Supports transfer downsizing; AHB data width can be the same as AXI or narrower
- Configurable synchronous or asynchronous AXI/AHB clock operations with any clock ratio



DW_axi_rs

AXI register slice

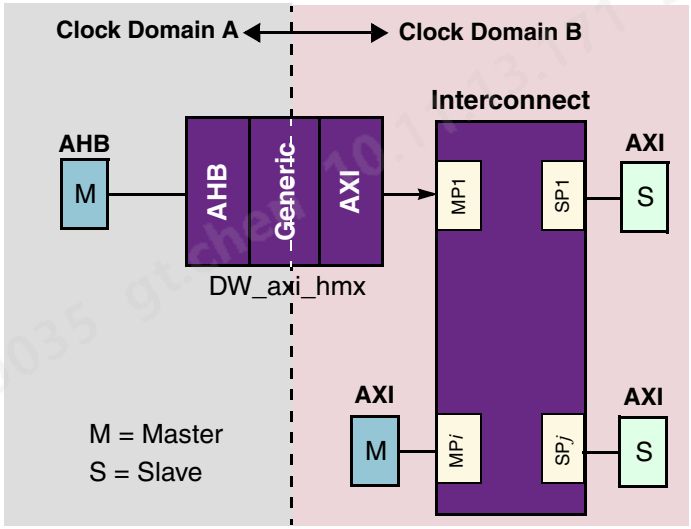
- Pipelines AXI channels to terminate critical timing paths
- Selected on a per AXI channel basis



DW_axi_hmx

This component is an interface module that connects an AHB master to an AXI subsystem.

- Single AHB master interface translated to AXI master interface
- Combine with a DW_ahb_eh2h to connect a full AHB subsystem or support endianness conversion
- Synchronous clock support, including slower AHB clock
- Support for locked transactions
- Low-power support

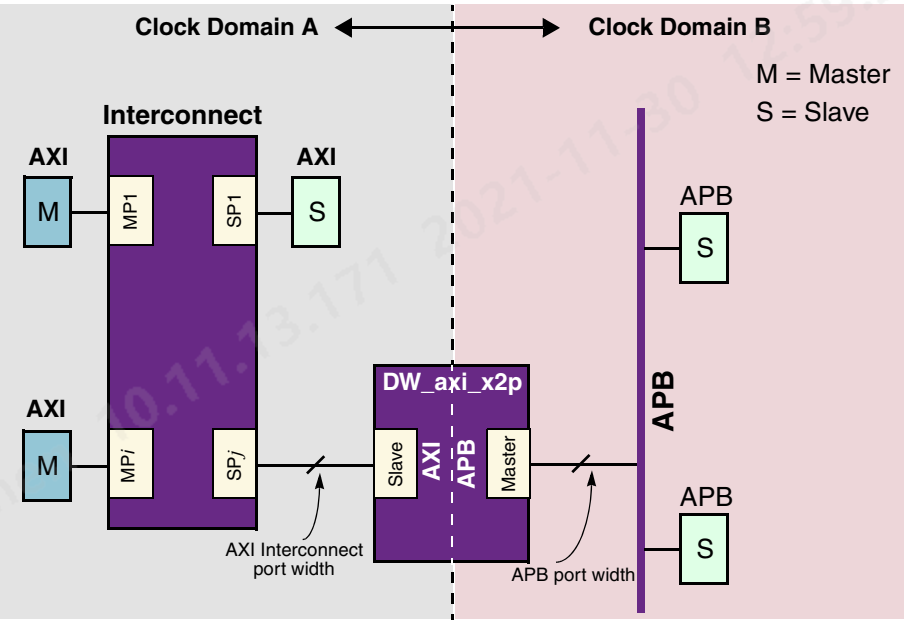


DW_axi_x2p

AXI-to-APB 3.0 Bridge

This bridge connects an AXI subsystem to an APB subsystem.

- Wait state support
- AMBA 3 and 2.0 compatible
- APB error response

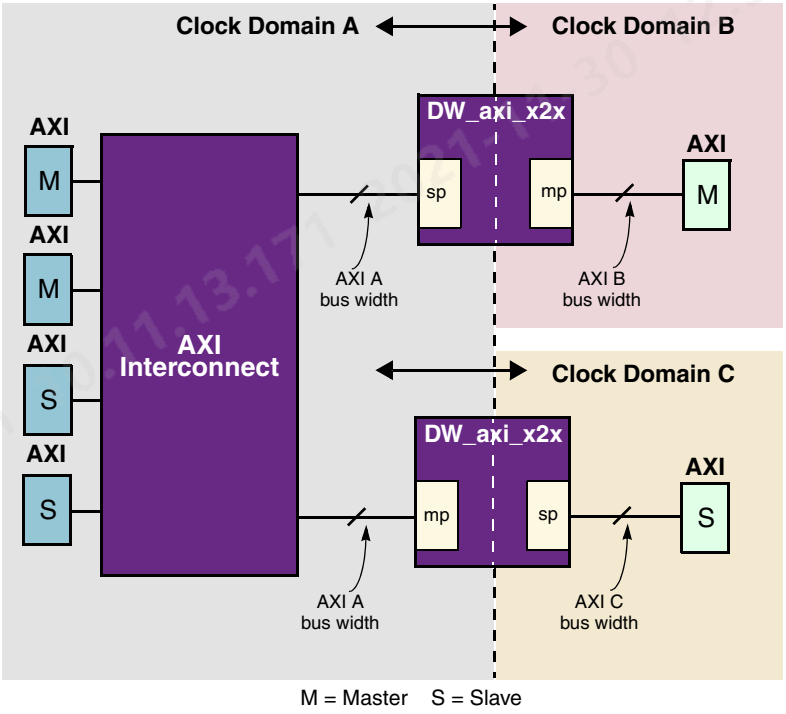


DW_axi_x2x

AXI-to-AXI 3.0 Bridge

This bridge connects an AXI master and an AXI slave, or it can connect incompatible AXI interconnect subsystems.

- AMBA 3 compatible
- Transparent operation
- Accepts simultaneous AXI transactions
- Configurable buffer depths



2 New Components

There are currently no AXI components scheduled for future release.

3 Application Notes

The following application notes have been developed to better help you begin using DesignWare AMBA 3 AXI components.

- "Connecting an AMBA 2.0 AHB Slave to an AMBA 3 AXI Subsystem"
- "Creating an AXI-to-APB 2.0 Bridge Using DesignWare IP for AMBA" on page 24
- "Connecting an AMBA 2.0 AHB Subsystem to an AMBA 3 AXI Subsystem" on page 30
- "Using DW_ahb_dmac in an AXI Subsystem" on page 37

Connecting an AMBA 2.0 AHB Slave to an AMBA 3 AXI Subsystem

With the transition of new designs to AMBA™ 3 AXI™, there is considerable existing AMBA 2.0 AHB-based intellectual property (IP) which continues to be very useful. To quickly integrate legacy AHB designs into AXI subsystems, without requiring time-consuming and risky re-design, Synopsys provides an efficient and reliable method of reuse.

This application note focuses on a specific example of connecting one AHB slave to an AXI subsystem, however, any AHB-based subsystem could be connected in a similar way. It reviews the steps required to integrate multiple DesignWare® Synthesizable IP using coreAssembler, as well as how to include non-DesignWare IP in the subsystem. Finally, it discusses various integration considerations, including how to configure the DesignWare Synthesizable IP used in this type of subsystem.

All DesignWare IP components used in this application note, as well as coreAssembler usage, are made available by a standard DesignWare Library license.

Example System Block Diagram

Figure 3 illustrates an example subsystem where an AXI interconnect connect to an AHB interconnect, using DW_axi_x2h.

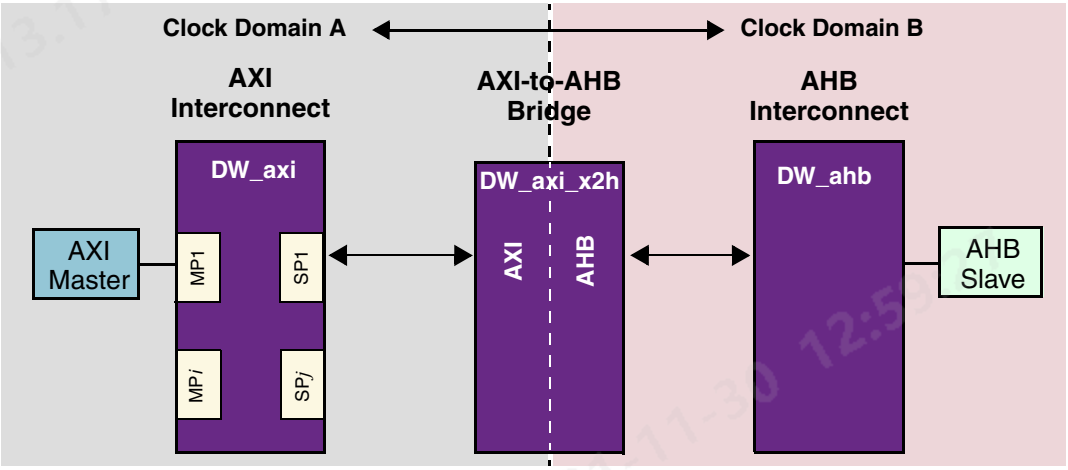


Figure 3: Example AHB Slave connected to AXI Subsystem

The components illustrated in Figure 3 on page 18 are described as follows:

- **AXI Master** – A component that generates new transactions to AXI slaves (for example, a microprocessor)
- **AXI Interconnect** (DW_axi) – DesignWare Synthesizable IP component that routes AXI requests/responses between AXI masters and AXI slaves
- **AXI-to-AHB Bridge** (DW_axi_x2h) – DesignWare Synthesizable IP component that processes AXI requests, translates them to the AHB bus and returns the AHB response to the AXI response channels
- **AHB Interconnect** (DW_ahb) – DesignWare Synthesizable IP component that is responsible for AHB bus arbitration, address decoding, and data multiplexing between AHB masters and AHB slaves
- **APB Slave** – A component that responds to transactions from DW_apb (for example, an interrupt controller or UART)

Creating the Subsystem

When you use the Synopsys coreAssembler tool with DesignWare Synthesizable IP, you can construct and simulate any single- or multi-layer AMBA-based subsystem that you can conceive. coreAssembler is a highly flexible, integrated and feature-rich design environment that allows you to select, configure, interconnect, simulate, and synthesize IP. This application note describes the specific coreAssembler steps required to connect an AHB slave to an AXI subsystem.

Required Tools, Components, and Licenses

- coreAssembler 5.1.1 or later (5.2 or later recommended)
- DesignWare Synthesizable IP for AMBA (*more information*)
  - DW_axi
  - DW_axi_x2h
  - DW_ahb
- DesignWare license

For detailed information about downloading and installing DesignWare Library Synthesizable IP, *click here*.

If you are not already familiar with using coreAssembler with DesignWare Synthesizable IP, *click here*. In that document, the tutorials in Chapter 6 provide step-by-step details of the process in the following subsection.

coreAssembler Steps

Once you have coreAssembler and DesignWare Library Synthesizable IP installed on your system, you can get started with creating your subsystem. In coreAssembler, create a new workspace, then complete the following instructions:

Add Subsystem Components and Interface Configuration

With any new coreAssembler workspace, you see a blank schematic, an Activity List, and a console for text input/output. The first activity to complete in the Activity List is Add Subsystem Components.

1. Insert components.

   The minimum set of components required are DW_axi, DW_axi_x2h, and DW_ahb. Use the menu item for **Schematic -> Add New Component**.

2. For each added component, right-click on each and use **Change Connection** to verify the interface connections are correct. Make sure the DW_axi_x2h component is connected to both DW_axi and DW_ahb. Any interface marked in red must be connected or marked as unused.

3. For each added component, right-click on each and use **Edit Interface Parameters** to verify the values of the parameters that affect that component's interfaces. Certain components (for example, DW_axi_x2h) do not have any interface parameters that are configurable, because their interfaces are automatically configured based on how attached components (for example, DW_axi and DW_ahb) are configured.

4. For any AXI masters or AHB slaves, if you are not using a DesignWare Synthesizable IP for those components, you can use **Export Interface**. You can export an AXI Master interface from DW_axi and an AHB Slave interface from DW_ahb. Export Interface creates top-level ports in your subsystem RTL for all signals in the interface, which you can then use to connect to the RTL for your AXI master and AHB slave RTL outside coreAssembler. Alternatively, with a coreAssembler license you can import your own IP RTL directly into coreAssembler using Import Component instead of Export Interface.

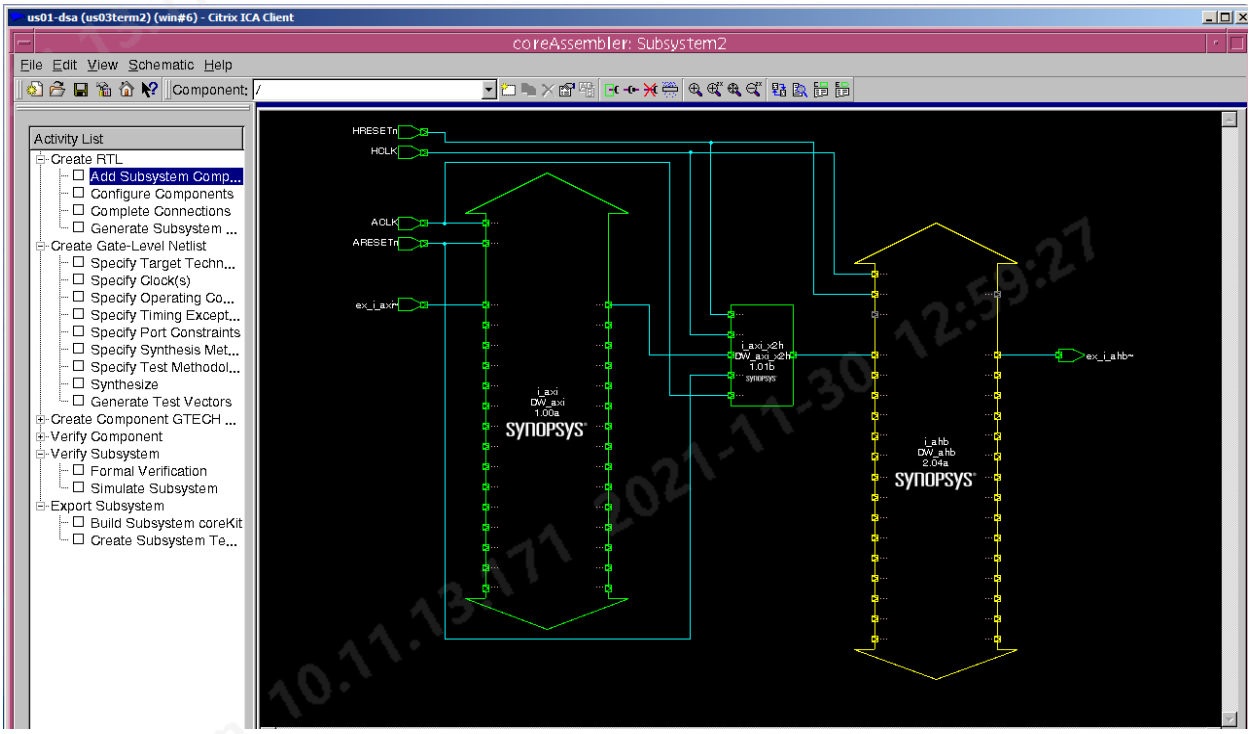


Figure 4: coreAssembler Example Subsystem

After completing these steps, click Apply in the lower right corner of the schematic. If you've configured a legal subsystem, the Add Subsystem Components activity will complete successfully. If the subsystem has errors, correct those errors and click Apply again.

Configure Components

To move to the next activity in coreAssembler, click on Configure Components in the Activity List. For this activity, you configure each individual component separately. For each component, if you are not sure what a specific configuration parameter controls, details on that parameter can be found in the databook for that component. The databook for every component in the subsystem is available from the **Help** menu. Another way to retrieve brief information on a parameter is to right-click on the parameter and choose **What's This?** from the menu.

Further discussion of specific configuration details for this subsystem are discussed in "Integration Considerations" later in this application note.

To complete this activity, click Apply in the lower right corner, or simply click on the next activity in the Activity List. When this activity completes, the RTL for each individual component, but not the top level of the subsystem, is written to the coreAssembler workspace directory.

Complete Connections

In the Complete Connections activity, you have the opportunity to manually connect/disconnect wires in the subsystem. In general, no manual changes are required. Complete this activity by clicking Apply or clicking on the next activity in the Activity List.

Generate Subsystem RTL

This is the last activity in the Create RTL activity group. Choose the RTL language you want the top level RTL to be written in and click Apply to complete the activity. When this activity completes processing, the configured RTL for the entire subsystem has been written to the coreAssembler workspace.

Additional Optional Activities in coreAssembler

- **Create Gate-Level Netlist** – This activity group is used to synthesize the subsystem. Support is provided for Design Compiler, DFT Compiler, Power Compiler, Physical Compiler, PrimeTime (timing model generation), and TetraMax (ATPG). Licenses are required for each tool used.
- **Verify Component** – For each DesignWare component, tests and component-specific testbenches are provided to demonstrate the functionality of the component. The tests and testbenches used in these activities are not reusable, however, the simulation waveforms are useful to examine detailed signal behavior.
- **Verify Subsystem** – The Formal Verification activity runs Formality (license required) on the entire synthesized subsystem. The Simulate Subsystem activity generates a custom testbench for your specific subsystem configuration and executes basic connectivity tests (also called “ping” tests). The testbench and tests created by this activity can be reused as a starter testbench for a larger subsystem or more detailed tests.
- **Create Component GTECH Simulation Model** – If VCS is the simulator used for the Subsystem Simulation or Verify Component activities, this activity group is not required. However, if a simulator other than VCS is selected, and you do not own a source license for the RTL of the DesignWare component, a GTECH simulation model is required and these activities must be completed before simulation can be run.
- **Export Subsystem** – If you intend to package the subsystem for re-use in different designs or to package your own subsystem IP for delivery to your customers, these activities can be used. They require a coreBuilder license. For more information, see the *coreBuilder User Guide*.

Integration Considerations

When integrating multiple DesignWare IP components, much of the details are handled automatically by coreAssembler:

- Component inputs/outputs that belong to standard interfaces (for example, the AXI protocol) are connected automatically.
- Component inputs/outputs that do not belong to standard interfaces (for example, interrupt pins) are automatically handled with default connections that are pre-defined by the DesignWare IP designer.
- Interface parameters (for example, AXI address bus width) are defined one time and then propagated to connected components automatically.
- Configured RTL generation for both components and the top level of the subsystem occurs automatically.
- Subsystem-level synthesis and verification are also highly automated.

For the specific configuration discussed in this application note, connecting an AHB slave to an AXI subsystem, there are a few integration considerations that deserve additional examination.

AMBA Lite

For the specific design example provided in this application note, if the connected AHB slave does not issue Split or Retry, the AHB subsystem could be configured as AMBA Lite, since the only connected AHB master is DW_axi_x2h.

To configure DW_ahb for AMBA Lite mode, the AMBA Lite interface parameter should be checked in the Add Subsystem Components activity. The parameter value chosen for AMBA Lite in DW_ahb automatically propagates to DW_axi_x2h. Both these components are optimized if the AMBA Lite mode parameter is enabled, resulting in smaller designs.

If more than one AHB master were connected, or if any connected AHB slave uses Split or Retry, then AMBA Lite mode could not be used.

Address Map

To ensure access for AXI masters to AHB slaves, take care in defining both the DW_axi and DW_ahb address maps. There is no address translation provided in DW_axi_x2h, so the address used in the AXI transaction is passed directly to the AHB bus.

In coreAssembler 5.2 or later, it is possible to allow coreAssembler to automatically initialize the address maps of both DW_axi and DW_ahb to a valid configuration. This initialization is enabled with a pop-up dialogue during the completion of the Add Subsystem Components activity. If any changes are made to the automatically initialized address map, carefully review all address maps before completing the Configure Components activity, to make sure all masters have access to appropriate slaves.

AXI/AHB Data Bus Widths

DW_axi_x2h requires the AXI data bus width to be greater than or equal to the width of the AHB data bus width. These interface parameters are defined on DW_axi and DW_ahb during the Add Subsystem Components activity. If the AHB data bus width is defined to be wider than the AXI data bus width, an error occurs when completing the Configure Components activity.

Discussion of how DW_axi_x2h adapts data widths between AXI and AHB can be found in Chapter 3 of the *DesignWare DW_axi_x2h Databook*, under Data Width Adaption.

Endianness

DW_axi_x2h currently only supports Little Endian on both the AXI and AHB buses. If DW_ahb is configured as Big Endian, an error occurs when completing the Add Subsystem Components activity.

AXI/AHB Lock Transactions

DW_axi_x2h does not support lock passing to the AHB bus by default, but this can be enabled by changing the configuration parameter X2H_PASS_LOCK. DW_axi supports for locked transactions is scheduled to be supported in version 1.02a of the component.

For detailed information on locking transaction support, see the databooks for the DW_axi, DW_axi_x2h, and DW_ahb components.

INCR Burst Support on AHB

For AHB buses with more than one AHB master, DW_ahb by default early burst terminates unspecified length INCR bursts from a lower-priority master, if a higher priority master requests the bus. The DW_axi_x2h supports using INCR bursts by default, and will correctly resume the remaining INCR transfers if interrupted by the DW_ahb.

This functionality is configurable in both the DW_ahb and the DW_axi_x2h. The DW_ahb can be configured to not interrupt unspecified length INCR bursts from a lower priority master (see DW_ahb parameter AHB_FULL_INCR). Also, the DW_axi_x2h can be configured to never use unspecified length INCR bursts (see DW_axi_x2h parameter X2H_USE_DEFINED_ONLY).

Reference Documentation

All of the following documents can be found in the *Guide to DesignWare AMBA IP Components Documentation*. This document can also be found on your local system, after installing the DesignWare Synthesizable IP for AMBA, at $DESIGNWARE_HOME/doc/amba/latest/intro.pdf.

- DW_axi_x2h Databook and Release Notes – For detailed information on how commands are transferred from AXI to AHB, refer to Chapter 3 (Functional Description) and Chapter 7 (Integration Considerations) in the DW_axi_x2h Databook.
- DW_axi Databook and Release Notes – Provides AXI interconnect details.
- DW_ahb Databook and Release Notes – Provides AHB interconnect details.
- Using coreAssembler with DesignWare Synthesizable IP – Chapter 6 provides tutorials for getting started with coreAssembler and DesignWare IP.
- *coreAssembler User Guide* – Provides general coreAssembler usage.
- AMBA 2.0 Specification – *Download here*.
- AMBA 3 Specification – *Download here*.

Creating an AXI-to-APB 2.0 Bridge Using DesignWare IP for AMBA

With the transition of new designs to AMBA™ 3 AXI™, there is considerable existing AMBA 2.0 APB-based intellectual property (IP) that continues to be very useful. Using Synopsys DesignWare® IP for AMBA and the coreAssembler tool, you can reliably integrate legacy APB 2.0 designs into AXI subsystems.

This application note covers how to connect any APB 2.0 subsystem to an AXI subsystem. It reviews the steps required to integrate multiple DesignWare Synthesizable IP using coreAssembler, as well as how to include non-DesignWare IP in the subsystem. Finally, it discusses various integration considerations, including how to configure the DesignWare Synthesizable IP used in this type of subsystem.

All DesignWare IP components used in this application note, as well as coreAssembler usage, are made available by a standard DesignWare Library license.

Example System Block Diagram

Figure 5 illustrates an example subsystem where an APB 2.0 subsystem is connected to an AXI subsystem.

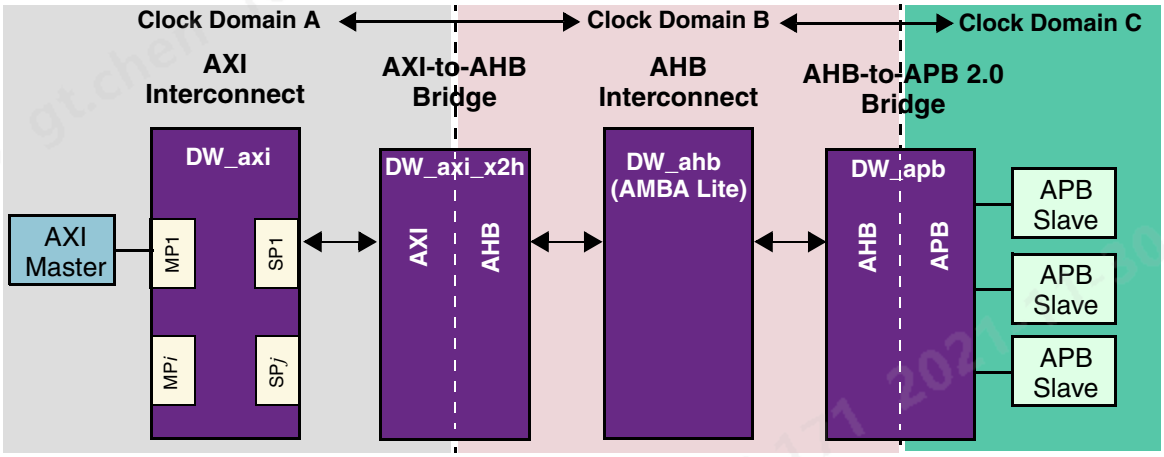


Figure 5: Example APB 2.0 Subsystem connected to AXI Subsystem

The components illustrated in Figure 5 are described as follows:

- **AXI Master** – A component that generates new transactions to AXI slaves (for example, a microprocessor)
- **AXI Interconnect** (DW_axi) – DesignWare Synthesizable IP component that routes AXI requests/responses between AXI masters and AXI slaves
- **AXI-to-AHB Bridge** (DW_axi_x2h) – DesignWare Synthesizable IP component that processes AXI requests, translates them to the AHB bus and returns the AHB response to the AXI response channels
- **AHB Interconnect** (DW_ahb) – DesignWare Synthesizable IP component that is responsible for AHB bus arbitration, address decoding, and data multiplexing between AHB masters and AHB slaves

- **AHB-to-APB 2.0 Bridge** (DW_apb) – DesignWare Synthesizable IP component that processes AHB requests, translates them to the APB bus and returns the APB response to the AHB bus
- **APB Slave** – A component that responds to transactions from DW_apb (for example, an interrupt controller or UART)

Creating the Subsystem

When you use the Synopsys coreAssembler tool with DesignWare Synthesizable IP, you can construct and simulate any single- or multi-layer AMBA-based subsystem that you can conceive. coreAssembler is a highly flexible, integrated and feature-rich design environment that allows you to select, configure, interconnect, simulate, and synthesize IP. This application note describes the specific coreAssembler steps required to connect an APB 2.0 subsystem to an AXI subsystem.

Required Tools, Components, and Licenses

- coreAssembler 5.1.1 or later (5.2 or later is recommended)
- DesignWare Synthesizable IP for AMBA (*more information*)
  - DW_axi
  - DW_axi_x2h
  - DW_ahb
  - DW_apb
  - DesignWare APB Peripherals (*more information*)
- DesignWare license

For detailed information about downloading and installing DesignWare Library Synthesizable IP, *click here*.

If you are not already familiar with using coreAssembler with DesignWare Synthesizable IP, *click here*. In that document, the tutorials in Chapter 6 provide step-by-step details of the process described in the following subsection.

coreAssembler Steps

Once you have coreAssembler and DesignWare Library Synthesizable IP installed on your system, you can get started with creating your subsystem. In coreAssembler, create a new workspace, then complete the following instructions:

Add Subsystem Components and Interface Configuration

With any new coreAssembler workspace, you see a blank schematic, an Activity List, and a console for text input/output. The first activity to complete in the Activity List is Add Subsystem Components.

1. Insert components. The minimum set of components required are DW_axi, DW_axi_x2h, DW_ahb, and DW_apb. Select **Schematic -> Add New Component**. Also, add one or more of the DesignWare APB Peripherals (for example, DW_apb_uart or DW_apb_i2c).
2. For each added component, right-click on each and use **Change Connection** to verify the interface connections are correct. Make sure the DW_axi_x2h component is connected to both DW_axi and DW_ahb. Any interface marked in red must be connected or marked as unused.

3. For each added component, right-click on each and use **Edit Interface Parameters** to verify the values of the parameters that affect that component's interfaces. Certain components (for example, DW_axi_x2h) do not have any interface parameters that are configurable, because their interfaces are automatically configured based on how attached components (for example, DW_axi and DW_ahb) are configured.
4. For any AXI masters, if you are not using a DesignWare Synthesizable IP for that component, you can use the Export Interface feature. To do this, right-click on DW_axi and choose **Export Interface**. In the dialogue that opens, make sure to select AXI Master from the component interface list. Export Interface creates top-level ports in your subsystem RTL for all signals in the interface, which you can use to connect to the RTL for your AXI master RTL outside coreAssembler. Alternatively, with a coreAssembler license you can import your own IP RTL directly into coreAssembler using Import Component instead of Export Interface.

   Similarly, if any APB slaves in your APB subsystem are not DesignWare Synthesizable IP, you can use Export Interface or Import Component for those, as well.

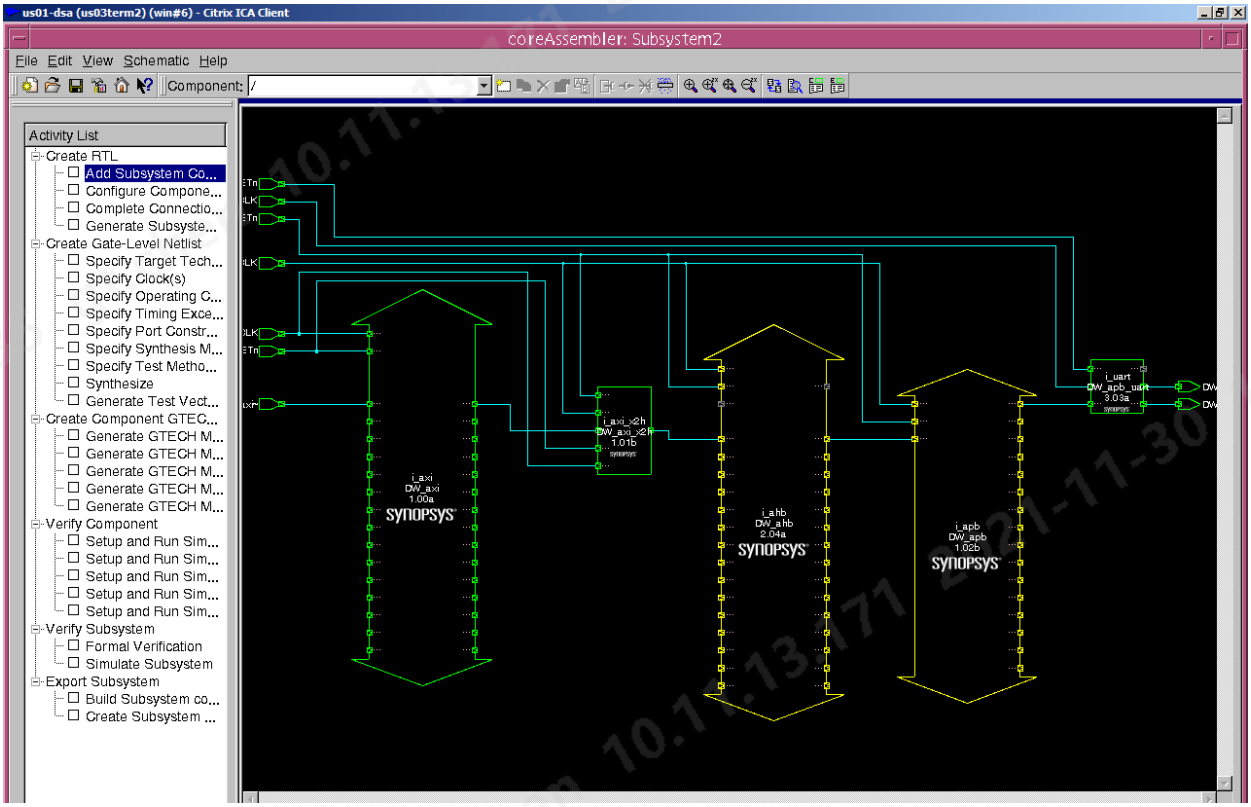


Figure 6: coreAssembler Example Subsystem

After completing these steps, click Apply in the lower right corner of the schematic. If you have configured a legal subsystem, the Add Subsystem Components activity completes successfully. If the subsystem has errors, correct those errors and click Apply again.

Configure Components

To move to the next activity in coreAssembler, click on Configure Components in the Activity List. For this activity, you configure each individual component separately. For each component, if you are not sure what a specific configuration parameter controls, details on that parameter can be found in the

databook for that component. The databook for every component in the subsystem is available from the **Help** menu. Another way to retrieve brief information on a parameter is to right-click on the parameter and choose **What's This?** from the menu.

Further discussion of specific configuration details for this subsystem are discussed in "Integration Considerations" later in this application note.

To complete this activity, click Apply in the lower right corner, or simply click on the next activity in the Activity List. When this activity completes, the RTL for each individual component, but not the top level of the subsystem, is written to the coreAssembler workspace directory.

Complete Connections

In the Complete Connections activity, you have the opportunity to manually connect/disconnect wires in the subsystem. In general, no manual changes are required. Complete this activity by clicking Apply or clicking on the next activity in the Activity List.

Generate Subsystem RTL

This is the last activity in the Create RTL activity group. Choose the RTL language you want the top level RTL to be written in and click Apply to complete the activity. When this activity completes processing, the configured RTL for the entire subsystem has been written to the coreAssembler workspace.

Additional Optional Activities in coreAssembler

- **Create Gate-Level Netlist** – This activity group is used to synthesize the subsystem. Support is provided for Design Compiler, DFT Compiler, Power Compiler, Physical Compiler, PrimeTime (timing model generation), and TetraMax (ATPG). Licenses are required for each tool used.
- **Verify Component** – For each DesignWare component, tests and component-specific testbenches are provided to demonstrate the functionality of the component. The tests and testbenches used in these activities are not reusable, however, the simulation waveforms are useful to examine detailed signal behavior.
- **Verify Subsystem** – The Formal Verification activity runs Formality (license required) on the entire synthesized subsystem. The Simulate Subsystem activity generates a custom testbench for your specific subsystem configuration and executes basic connectivity tests (also called "ping" tests). The testbench and tests created by this activity can be reused as a starter testbench for a larger subsystem or more detailed tests.
- **Create Component GTECH Simulation Model** – If VCS is the simulator used for the Subsystem Simulation or Verify Component activities, this activity group is not required. However, if a simulator other than VCS is selected, and you do not own a source license for the RTL of the DesignWare component, a GTECH simulation model is required and these activities must be completed before simulation can be run.
- **Export Subsystem** – If you intend to package the subsystem for re-use in different designs or to package your own subsystem IP for delivery to your customers, these activities can be used. They require a coreBuilder license. For more information, see the *coreBuilder User Guide*.

Integration Considerations

When integrating multiple DesignWare IP components, coreAssembler automatically handles much of the details:

- Component inputs/outputs that belong to standard interfaces (for example, the AXI protocol) are connected automatically.

- Component inputs/outputs that do not belong to standard interfaces (for example, interrupt pins) are automatically handled with default connections that are pre-defined by the DesignWare IP designer.
- Interface parameters (for example, AXI address bus width or APB data bus width) are defined once and then propagated to connected components automatically.
- Configured RTL generation for both components and the top level of the subsystem occurs automatically.
- Subsystem-level synthesis and verification are also highly automated.

For the specific configuration discussed in this application note—connecting an APB 2.0 subsystem to an AXI subsystem—there are a few integration considerations that deserve additional examination. These topics primarily cover DW_axi_x2h, the AXI-to-AHB bridge.

AMBA Lite

For the specific design example provided in this application note, if the only slave connected to DW_ahb is DW_apb, the AHB subsystem could be configured as AMBA Lite because the only connected AHB master is DW_axi_x2h. DW_apb does not issue Split or Retry to the AHB bus, so AMBA Lite mode is possible. If additional AHB slaves are connected to DW_ahb, as long as they do not issue Split or Retry, the AHB can still be configured as AMBA Lite.

To configure DW_ahb for AMBA Lite mode, the AMBA Lite interface parameter should be checked in the Add Subsystem Components activity. The parameter value chosen for AMBA Lite in DW_ahb automatically propagates to DW_axi_x2h. Both these components are optimized if the AMBA Lite mode parameter is enabled, resulting in smaller designs.

If more than one AHB master were connected, or if any connected AHB slave uses Split or Retry, then AMBA Lite mode can not be used.

Address Map

To ensure access for AXI masters to APB slaves, take care in defining both the DW_axi, DW_ahb, and DW_apb address maps. There is no address space translation provided in either DW_axi_x2h or DW_apb, so the address used in the AXI transaction is passed directly to the APB bus.

In coreAssembler 5.2 or later, it is possible to allow coreAssembler to automatically initialize the address maps of DW_axi, DW_ahb, and DW_apb to a valid configuration. This initialization is enabled with a pop-up dialogue during the completion of the Add Subsystem Components activity. If any changes are made to the automatically initialized address map, carefully review all address maps before completing the Configure Components activity, to make sure all masters have access to appropriate slaves.

AXI/AHB/APB Data Bus Widths

DW_axi_x2h requires the AXI data bus width to be greater than or equal to the width of the AHB data bus width. These interface parameters are defined on DW_axi and DW_ahb during the Add Subsystem Components activity. If the AHB data bus width is defined to be wider than the AXI data bus width, an error occurs when completing the Configure Components activity.

Discussion of how DW_axi_x2h adapts data widths between AXI and AHB can be found in Chapter 3 of the *DesignWare DW_axi_x2h Databook*, under Data Width Adaption.

The DW_apb data bus width must be 32, 16, or 8 bits, so it is impossible to create a configuration where the APB data bus is wider than the AXI data bus. The *DesignWare DW_axi_x2h Databook*, Chapter 7, details how reads/writes are handled when the APB data bus width is equal to or smaller than the AHB data bus width.

Endianness

DW_axi_x2h currently only supports Little Endian on both the AXI and AHB buses. If DW_ahb is configured as Big Endian, an error occurs when completing the Add Subsystem Components activity. APB subsystems using DW_apb are always Little Endian.

INCR Burst Support on AHB

Although DW_apb does not monitor the HBURST signal of the AHB bus, burst transactions from DW_axi_x2h are supported. DW_apb treats each read or write beat (of a multi-beat burst) as a single APB transaction; therefore, AHB burst requests are no different than AHB SINGLE requests.

For AHB buses with more than one AHB master, by default DW_ahb early burst terminates unspecified length INCR bursts from a lower-priority master, if a higher priority master requests the bus. DW_axi_x2h supports using INCR bursts by default and correctly resumes the remaining INCR transfers if interrupted by DW_ahb.

This functionality is configurable in both DW_ahb and DW_axi_x2h. DW_ahb can be configured to not interrupt unspecified length INCR bursts from a lower priority master (see the DW_ahb parameter AHB_FULL_INCR). Also, DW_axi_x2h can be configured to never use unspecified length INCR bursts (see the DW_axi_x2h parameter X2H_USE_DEFINED_ONLY).

Reference Documentation

All of the following documents can be found in the *Guide to DesignWare AMBA IP Components Documentation*. This document can also be found on your local system, after installing the DesignWare Synthesizable IP for AMBA, at $DESIGNWARE_HOME/doc/amba/latest/intro.pdf.

- DW_apb Databook and Release Notes – Provides AHB-to-APB 2.0 bridge details.
- Using coreAssembler with DesignWare Synthesizable IP – Chapter 6 provides tutorials for getting started with coreAssembler and DesignWare IP.
- *coreAssembler User Guide* – Provides general coreAssembler usage.
- AMBA 2.0 Specification – *Download here*.
- AMBA 3 Specification – *Download here*.

Connecting an AMBA 2.0 AHB Subsystem to an AMBA 3 AXI Subsystem

With the transition of new designs to AMBA 3 AXI, there is considerable existing AMBA 2.0 AHB-based intellectual property (IP) which can continue to be very useful. To quickly integrate legacy AHB designs into AXI subsystems, without requiring time-consuming and risky re-design, Synopsys provides an efficient and reliable method of reuse.

The DW_axi_x2h provides a method to transfer AXI generated transactions to an AHB bus. It provides the subsystem designer with an easy way to reuse existing AHB components or full AHB subsystems in new designs that use the higher performance AXI bus. For more details, please refer to application note "Connecting an AMBA 2.0 AHB Slave to an AMBA 3 AXI Subsystem" on page 17.

The DW_axi_hmx enables a single AHB master to be connected directly to an AXI bus. The focus of this application note is to show how you can connect an AHB-based subsystem to an AXI subsystem using DW_ahb_eh2h and DW_axi_hmx. So, a complete bridging solution using DesignWare Synthesizable IP is available between AHB and AXI subsystems.

This application note reviews the steps required to integrate multiple DesignWare Synthesizable IP using coreAssembler, as well as how to include non-DesignWare IP in the subsystem. Finally, it discusses various integration considerations, including how to configure DesignWare Synthesizable IP used in this type of subsystem.

All DesignWare components referenced in this application note, as well as the Synopsys coreAssembler, are made available with a standard DesignWare license.

Example System Block Diagram

Figure 7 illustrates a subsystem in which an AHB interconnect connects to an AXI interconnect using DW_ahb_eh2h and DW_axi_hmx.

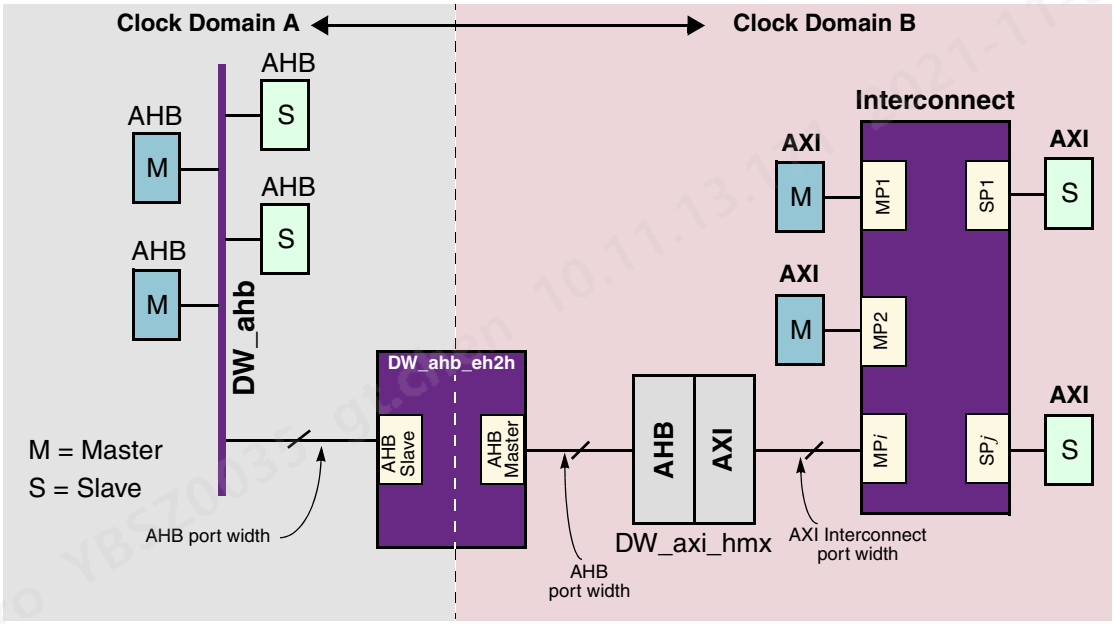


Figure 7: Example AHB-to-AXI Bridge

In this bridge solution, DW_ahb_eh2h performs the following functions:

- Downsizing of data for differing AHB/AXI data widths
- Support for synchronous or asynchronous clock domains
- Off-loading of the AHB bus when a transaction on the AXI bus is ongoing

The DW_axi_hmx performs the following functions:

- Support for endianness conversion
- Protocol conversion

The components in this subsystem are:

AHB Master	A component that delivers new transactions to AHB slaves (for example, a microprocessor)
AHB Interconnect (DW_ahb)	DesignWare Synthesizable IP component responsible for AHB bus arbitration, address decoding, and data muxing between AHB masters and AHB slaves
AHB-to-AHB Bridge (DW_ahb_eh2h)	DesignWare Synthesizable IP component that processes AXI requests, translates them to the AHB bus and returns the AHB response to the AXI response channels
AHB Interconnect (DW_ahb)	DesignWare Synthesizable IP component that establishes a communication link between two AHB subsystems, allowing for data exchange between a primary master and a secondary slave. It is attached as a slave to first AHB subsystem and as a master to the second AHB subsystem.
AHB-to-AXI Gasket (DW_axi_hmx)	DesignWare Synthesizable IP component that connects an AHB master to an AXI subsystem
AXI Interconnect (DW_axi)	DesignWare Synthesizable IP component that routes AXI requests/responses between AXI masters and AXI slaves
AXI Slave	A component that responds to transactions from AXI masters

Creating the Subsystem

The Synopsys coreAssembler tool is a highly flexible, integrated and feature-rich design environment that allows you to select, configure, interconnect, simulate, and synthesize IP. By using the coreAssembler tool with DesignWare Synthesizable IP, you can construct and simulate any single- or multi-layer AMBA-based subsystem.

This application note describes the coreAssembler steps required to connect an AHB-based subsystem to an AXI subsystem.

Required Tools, Components and Licenses

- coreAssembler 2006.03-SP1
- DesignWare Synthesizable IP for AMBA (*more information*)
  - DW_axi
  - DW_axi_hmx
  - DW_ahb_eh2h

  - DW_ahb
- DesignWare license

For detailed information about downloading and installing DesignWare Library Synthesizable IP, *click here*.

If you are not already familiar with using coreAssembler with DesignWare Synthesizable IP, *click here*. In that document, the tutorials in Chapter 6 provide step-by-step details of the process described in the following subsection.

coreAssembler Steps

Once coreAssembler and DesignWare Library Synthesizable IP are installed on your system, you can start creating your subsystem. In coreAssembler, create a new workspace, then follow these instructions:

Add Subsystem Components and Interface Configuration

With any new coreAssembler workspace, you will see a blank schematic, an Activity List, and a console for text input/output. The first activity to complete in the Activity List is Add Subsystem Components:

1. Insert components.

   The minimum set of components required are DW_ahb, DW_axi_eh2h, DW_axi_hmx and DW_axi. Use the menu item for **Schematic > Add New Component**.

2. For each added component, right-click on each and use **Change Connection** to verify the interface connections are correct. Any interface marked in red must be connected or marked as unused.

3. For each added component, right-click on each and use **Edit Interface Parameters** to verify the values of the parameters that affect that component's interfaces. Certain components (for example, DW_axi_eh2h) do not have any interface parameters that are configurable, since their interfaces are automatically configured based on how attached components (for example, DW_ahb and DW_axi_hmx) are configured.

4. For any AHB master or AXI slave components that are not DesignWare Synthesizable IP, you can use **Export Interface**. You can export an AHB Master interface from DW_ahb and an AXI Slave interface from DW_axi. **Export Interface** creates top-level ports in your subsystem RTL for all signals in the interface, which you can then use to connect to the RTL for your AHB master and AXI slave RTL outside of the coreAssembler environment. Alternatively, with a coreAssembler license you can import your own IP RTL directly into coreAssembler using **Import Component**.

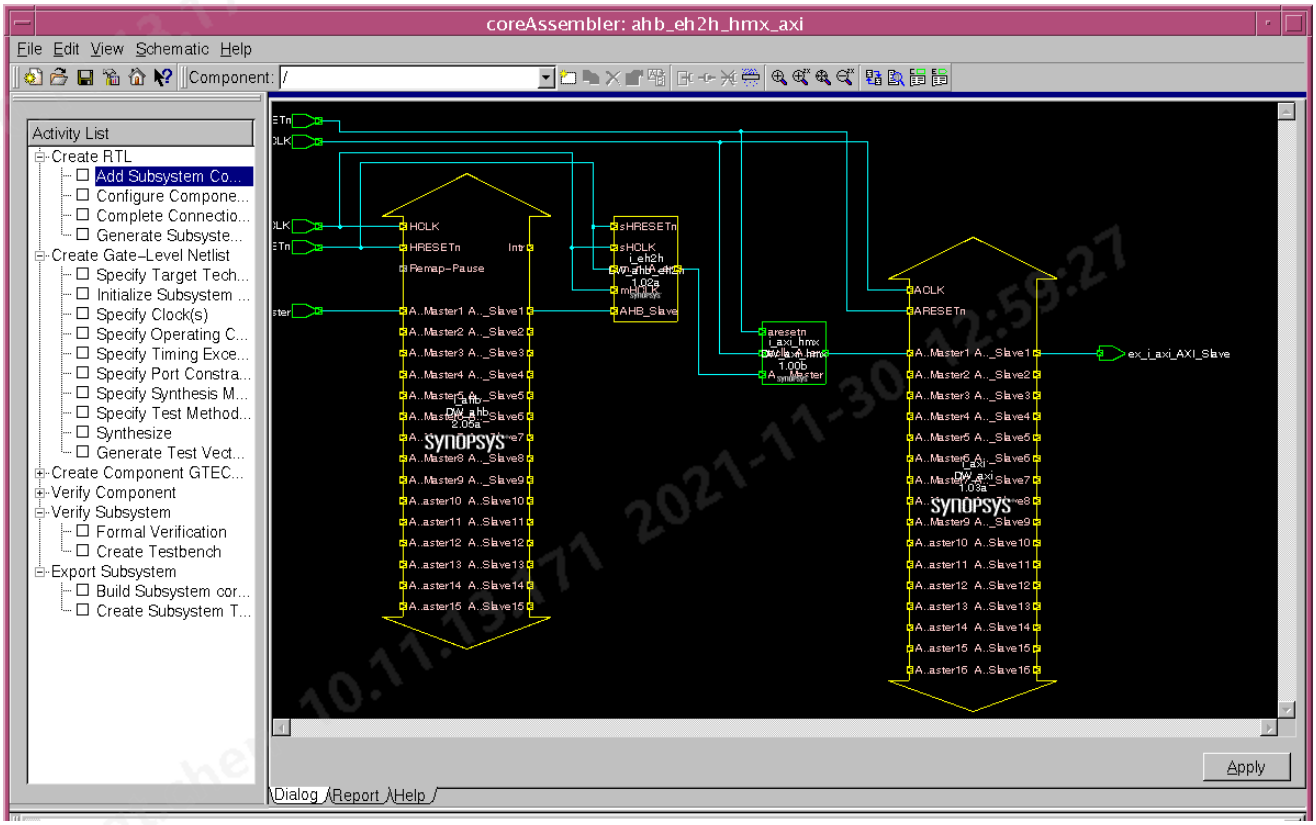


Figure 8: coreAssembler Example Subsystem

After completing these steps, click Apply in the lower right corner of the schematic. If you've configured a legal subsystem, the Add Subsystem Components activity will complete successfully. If the subsystem has errors, correct them and click Apply again.

Configure Components

The next step in coreAssembler is to click on **Configure Components** in the Activity List. For this activity, you will configure each individual component separately. For each component, if you are not sure what a specific configuration parameter controls, details can be found in the databook for that component. The databook for every component in the subsystem is available from the **Help** menu. Another way to retrieve brief information on a parameter is to right-click on the parameter and choose **What's This?** from the menu.

Further discussion of specific configuration details for this subsystem are discussed in "Integration Considerations" on page 34.

To complete this activity, click Apply in the lower right corner, or simply click on the next activity in the Activity List. When the activity is complete, the RTL for each individual component, but not for the top level of the subsystem, is written to the coreAssembler workspace directory.

Complete Connections

In the Complete Connections activity, you have the opportunity to manually connect/disconnect wires in the subsystem. Generally, no manual changes are required. Complete this activity by clicking Apply or click on the next activity in the Activity List.

Generate Subsystem RTL

This is the last activity in the Create RTL activity group. Choose the RTL language you want the top level RTL to be written in and click Apply to complete the activity. When this activity completes processing, the configured RTL for the entire subsystem has been written to the coreAssembler workspace.

Additional Optional Activities in coreAssembler

- **Create Gate-Level Netlist** – This activity group is used to synthesize the subsystem. Support is provided for Design Compiler, DFT Compiler, Power Compiler, Physical Compiler, PrimeTime (timing model generation), and TetraMax (ATPG). Licenses are required for each tool used.
- **Verify Component** – For each DesignWare component, tests and component-specific testbenches are provided to demonstrate the functionality of the component. The tests and testbenches used in these activities are not reusable, however the simulation waveforms are useful to examine detailed signal behavior.
- **Verify Subsystem** – The Formal Verification activity runs Formality (license required) on the entire synthesized subsystem. The Simulate Subsystem activity generates a custom testbench for your specific subsystem configuration and executes basic connectivity tests (also called “ping” tests). The testbench and tests created by this activity can be reused as a starter testbench for a larger subsystem or more detailed tests.
- **Create Component GTECH Simulation Model** – If VCS is the simulator used for the Subsystem Simulation or Verify Component activities, this activity group is not required. However, if a simulator other than VCS is selected, and you don't own a source license for the RTL of the DesignWare component, a GTECH simulation model is required and these activities must be completed before simulation can be run.
- **Export Subsystem** – If you intend to package the subsystem for re-use in different designs, or to package your own subsystem IP for delivery to your customers, these activities can be used. They require a coreBuilder license. For more information, see the *coreBuilder User Guide*.

Integration Considerations

When integrating multiple DesignWare IP components, much of the detail is handled automatically by coreAssembler:

- Component inputs/outputs that belong to standard interfaces (for example, the AXI protocol) are connected automatically
- Component inputs/outputs that do not belong to standard interfaces (for example, interrupt pins) are automatically handled with default connections that are pre-defined by the DesignWare IP designer
- Interface parameters (for example, AXI address bus width) are defined once and then propagated to connected components automatically
- Configured RTL generation for both components and the top level of the subsystem occurs automatically
- Subsystem-level synthesis and verification are also highly automated.

For the specific configuration discussed in this application note, connecting an AHB-based subsystem to an AXI subsystem, there are several integration considerations that deserve further examination.

Address Map

In coreAssembler 5.2 or later, it is possible to allow coreAssembler to automatically initialize the address maps of both the DW_axi and DW_ahb to a valid configuration. This initialization is enabled with a pop-up dialogue during the completion of the Add Subsystem Components activity. If any changes are made to the automatically initialized address map, carefully review all address maps before completing the Configure Components activity to make sure all masters have access to appropriate slaves.

AXI/AHB Address and Data Bus Widths

The DW_axi_hmx requires the AXI address bus width to be greater than or equal to the width of the AHB address bus width. These interface parameters are defined on the DW_axi and DW_ahb during the Add Subsystem Components activity. If the AHB address bus width is defined to be wider than the AXI address bus width, an error will occur when completing the Configure Components activity.

The data bus width conversion occurs in DW_ahb_eh2h. DW_axi_hmx supports same data bus width on both AHB and AXI interfaces.

Endianness

The DW_axi_hmx provides support for data endianness conversion between the AHB and AXI. The endianness of both the AHB and AXI buses is configured by setting two hardware configuration parameters, HMX_ENDIAN_AHB and HMX_ENDIAN_CONVERT. There is no need to perform any conversion in DW_ahb_eh2h.

To see how DW_axi_hmx provides support for data endianness conversion between AXI and AHB, see Chapter 3 of the *DesignWare DW_axi_hmx Databook*, under “Endianness Conversion.”

Burst Length Translation

All AHB hburst types, except one are translated to the corresponding AXI arburst/awburst types of length arlen/awlen. The one exception is AHB INCR, incrementing bursts of undefined length. This type is translated into AXI SINGLE transactions.

Reference Documentation

The following documents can be found in the *Guide to DesignWare AMBA IP Components Documentation*. Upon installation of the DesignWare Synthesizable IP for AMBA, this document can also be found on your local system at $DESIGNWARE_HOME/doc/amba/latest/intro.pdf.

- DW_axi_hmx Databook and Release Notes – Detailed information on how commands are transferred from AHB master to an AXI Subsystem. See databook Chapter 3 (Functional Description) and Chapter 7 (Integration Considerations).
- DW_ahb_eh2h Databook and Release Notes – Detailed information on how commands are transferred from primary AHB to secondary AHB subsystem. See databook Chapter 3 (Functional Description) and Chapter 9 (Integration Considerations).
- DW_axi Databook and Release Notes – AXI interconnect details.
- DW_ahb Databook and Release Notes – AHB interconnect details.
- DW_ahb_dmac Databook and Release Notes – Provides AHB DMA Controller details.
- Using coreAssembler with DesignWare Synthesizable IP – Chapter 6 provides tutorials for getting started with coreAssembler and DesignWare IP.

- *coreAssembler User Guide* – Guide to general coreAssembler usage.
- AMBA 2.0 Specification – *Download here*.
- AMBA 3 Specification – *Download here*.

Using DW_ahb_dmac in an AXI Subsystem

The AHB DMA Controller (DW_ahb_dmac) can be used in an AHB subsystem to perform DMA transfers between AHB peripherals. When the AHB subsystem is bridged to an AXI subsystem through a combination of DW_ahb_eh2h and DW_axi_hmx, it is possible to do DMA transfers between AHB and AXI peripherals. To learn how to connect an AHB subsystem to an AXI subsystem (using a combination of DW_ahb_eh2h and DW_axi_hmx), refer to the application note, "Connecting an AMBA 2.0 AHB Subsystem to an AMBA 3 AXI Subsystem" on page 30.

For transfers between AXI peripherals, the DW_ahb_dmac can be connected directly to an AXI subsystem.This application note describes how to connect DW_ahb_dmac to an AXI subsystem to allow DMA transfers between AXI peripherals. It describes how to use Synopsys coreAssembler to integrate multiple DesignWare Synthesizable IP, as well as how to incorporate non-DesignWare IP into a subsystem. Finally, it discusses various integration considerations, including how to configure DesignWare Synthesizable IP used in this type of subsystem.

All DesignWare components used in this application note, as well as coreAssembler usage, are made available via a standard DesignWare license.

Example System Block Diagrams

Figure 9 and Figure 10 illustrate the use of the DMA controller (DW_ahb_dmac) in two different AXI subsystem configurations.

Subsystem Example 1

Figure 9 illustrates a subsystem configured with DW_ahb_dmac on an AXI bus and using DW_axi_x2h and DW_axi_hmx components. In this case, DW_ahb_dmac is being programmed from an AXI master through an AXI-to-AHB bridge.

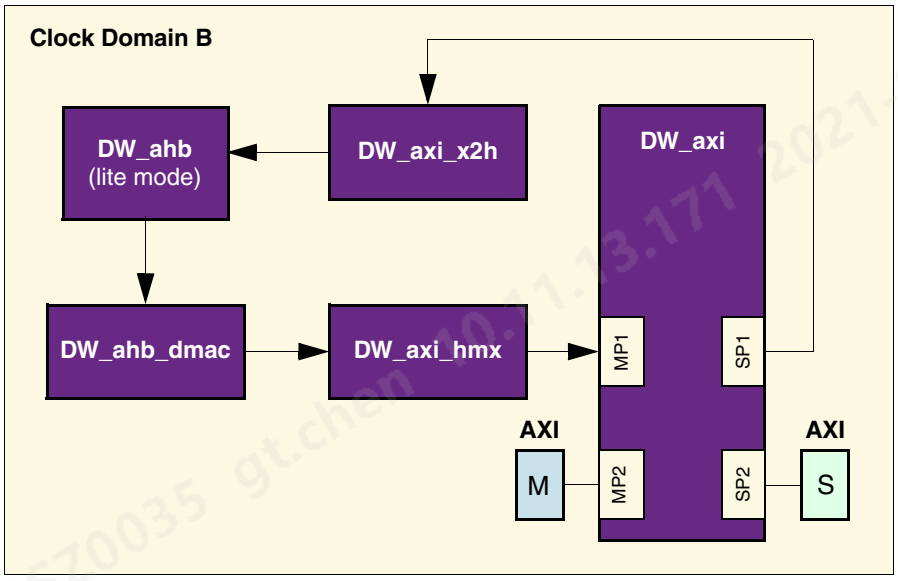


Figure 9: DW_ahb_dmac in an AXI Subsystem Using DW_axi_x2h and DW_axi_hmx

The components in this subsystem are:

AXI Master	A component that generates new transactions to AXI slaves (for example, a microprocessor)
AXI Interconnect (DW_axi)	DesignWare Synthesizable IP component that routes AXI requests/responses between AXI masters and AXI slaves
AXI-to-AHB Bridge (DW_axi_x2h)	DesignWare Synthesizable IP component that processes AXI requests, translates them to the AHB bus and returns the AHB response to the AXI response channels
AHB Interconnect (DW_ahb)	DesignWare Synthesizable IP component that is responsible for AHB bus arbitration, address decoding and data multiplexing between AHB masters and AHB slaves
AHB DMA Controller (DW_ahb_dmac)	DesignWare Synthesizable IP component that transfers data from a source peripheral to a destination peripheral over one or more AHB bus
AHB-to-AXI Gasket (DW_axi_hmx)	DesignWare Synthesizable IP component that connects an AHB master to an AXI Subsystem
AXI Slave	A component that responds to transactions from AXI masters

Subsystem Example 2

Figure 10 illustrates a subsystem in which DW_ahb_dmac connects to an AXI interconnect through the DW_axi_hmx. In this case, DW_ahb_dmac is being programmed from an AHB master through an AHB subsystem.

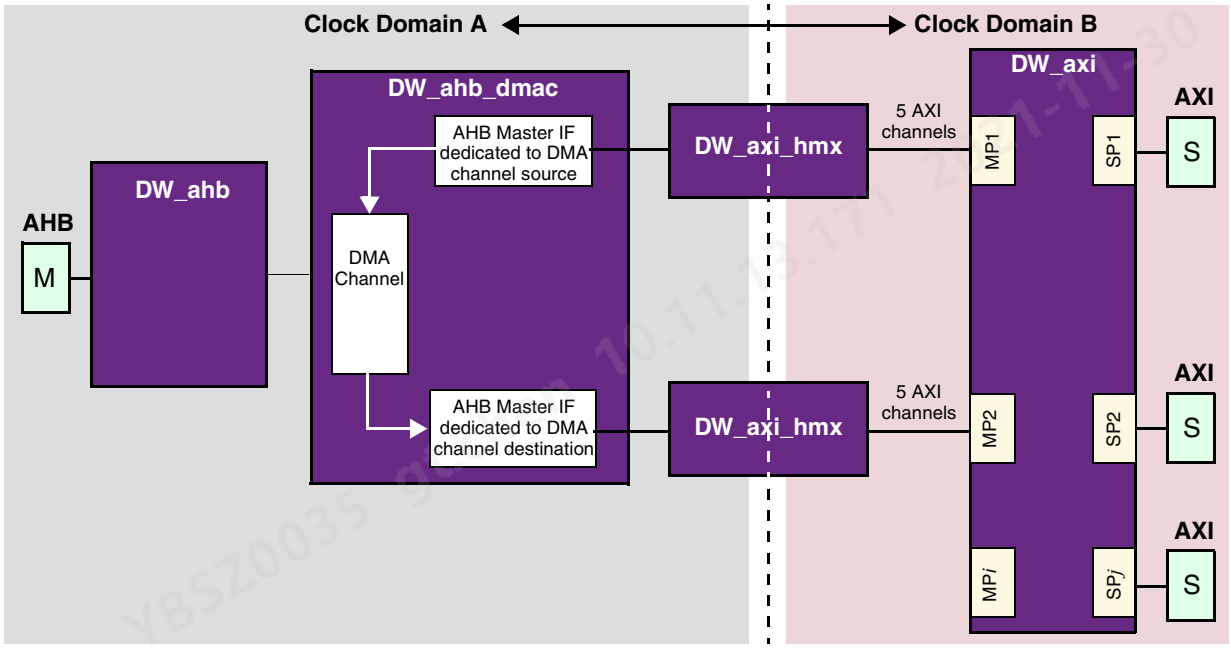


Figure 10: DW_ahb_dmac Connected in an AXI Subsystem Using DW_axi_hmx

This example uses two dedicated AXI master interfaces, one for the source peripheral and the other for the destination peripheral, to obtain higher DMA performance.

The components in this subsystem are:

AHB Master	A component that generates transactions to AHB slaves
AHB Interconnect (DW_ahb)	DesignWare Synthesizable IP component that is responsible for AHB bus arbitration, address decoding and data multiplexing between AHB masters and AHB slaves
AHB DMA Controller (DW_ahb_dmac)	DesignWare Synthesizable IP component that transfers data from a source peripheral to a destination peripheral over one or more AHB bus
AHB-to-AXI Gasket (DW_axi_hmx)	DesignWare Synthesizable IP component that connects an AHB master to an AXI Subsystem
AXI Interconnect (DW_axi)	DesignWare Synthesizable IP component that routes AXI requests/responses between AXI masters and AXI slaves
AXI Slave	A component that responds to transactions from AXI masters

Note

Clock domains A and B have to be synchronous to each other in this case. For asynchronous clocks, refer to the application note ""Connecting an AMBA 2.0 AHB Subsystem to an AMBA 3 AXI Subsystem" on page 30."

Creating the Subsystem

When you use the Synopsys coreAssembler tool with DesignWare Synthesizable IP, you can construct and simulate any conceivable single- or multi-layer AMBA-based subsystem. coreAssembler is a highly flexible, integrated and feature-rich design environment that allows you to select, configure, interconnect, simulate, and synthesize IP. This application note describes the specific coreAssembler steps to connect DW_ahb_dmac to an AXI subsystem as illustrated in Figure 10 on page 38.

Required Tools, Components and Licenses

- coreAssembler 2006.03-SP1
- DesignWare Synthesizable IP for AMBA (*more information*)
  - DW_ahb
  - DW_ahb_dmac
  - DW_axi_hmx
  - DW_axi
- DesignWare license

For detailed information about downloading and installing DesignWare Library Synthesizable IP, *click here*.

If you are not already familiar with using coreAssembler with DesignWare Synthesizable IP, *click here*. In that document, the tutorials in Chapter 6 provide step-by-step details of the process described in the following subsection.

coreAssembler Steps

Once coreAssembler and DesignWare Library Synthesizable IP are installed on your system, you can start creating your subsystem. In coreAssembler, create a new workspace, then proceed as follows:

Add Subsystem Components and Interface Configuration

With any new coreAssembler workspace, you will see a blank schematic, an Activity List, and a console for text input/output. The first activity to complete in the Activity List is Add Subsystem Components:

1. Insert components.

   The minimum set of components required are DW_axi, DW_ahb, DW_ahb_dmac and two instances each of DW_axi_hmx. Use the menu item for **Schematic > Add New Component**.

2. For each added component, right-click on each and use **Edit Interface Parameters** to verify the values of the parameters that affect that component's interfaces. For DW_ahb_dmac, set DMAH_NUM_MASTER_INT, the number of AHB master interfaces, to 2.

3. For each added component, right-click on each and use **Change Connection** to verify the interface connections are correct. Any interface marked in red must be connected or marked as unused. For the second DW_axi_hmx instance, connect its AHB master interface to DW_ahb_dmac's master 2 interface.

4. For any AHB masters, if you are not using a DesignWare Synthesizable IP for that component, you can use **Export Interface**. To do this, right-click on DW_ahb and choose **Export Interface**. In the dialogue that opens, make sure to select **AHB Master** from the component interface list. **Export Interface** creates top-level ports in your subsystem RTL for all signals in the interface, which you can then use to connect to the RTL for your AHB master RTL outside coreAssembler. Alternatively, with a coreAssembler license you can import your own IP RTL directly into coreAssembler using **Import Component**, and not use Export Interface.

Similarly, if any AXI slaves in your AXI subsystem are not DesignWare Synthesizable IP you can use Export Interface or Import Component for those, as well.

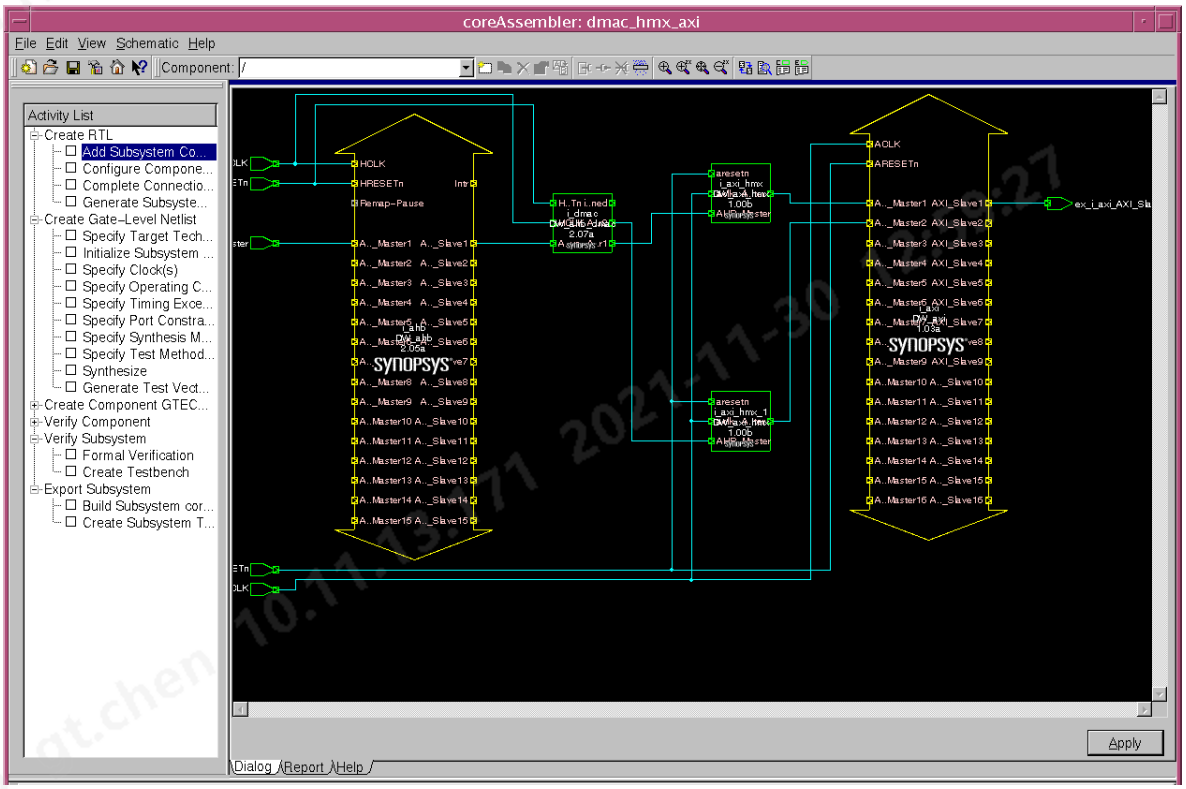


Figure 11: coreAssembler Example Subsystem

After completing these steps, click Apply in the lower right corner of the schematic. If you've configured a legal subsystem, the **Add Subsystem Components** activity will complete successfully. If the subsystem contains errors, correct those errors and click Apply again.

Configure Components

To move to the next activity in coreAssembler, click on Configure Components in the Activity List. For this activity, you configure each individual component separately. For each component, if you are not sure what a specific configuration parameter controls, details on that parameter can be found in the databook for that component. The databook for each component in the subsystem is available from the **Help** menu. Another way to retrieve brief information on a parameter is to right-click on the parameter and choose **What's This?** from the menu.

To complete this activity, click Apply in the lower right corner, or simply click on the next activity in the Activity List. When this activity completes, the RTL for each individual component, but not the top level of the subsystem, is written to the coreAssembler workspace directory.

Complete Connections

In the Complete Connections activity, you have the opportunity to manually connect/disconnect wires in the subsystem. Typically, no manual changes are required. Complete this activity by clicking Apply or clicking on the next activity in the Activity List.

Generate Subsystem RTL

This is the last activity in the Create RTL activity group. Choose the RTL language in which you want the top level RTL to be written, and click Apply to complete this activity. When the activity completes processing, the configured RTL for the entire subsystem has been written to the coreAssembler workspace.

Additional Optional Activities in coreAssembler

- **Create Gate-Level Netlist** – This activity group is used to synthesize the subsystem. Support is provided for Design Compiler, DFT Compiler, Power Compiler, Physical Compiler, PrimeTime (timing model generation), and TetraMax (ATPG). Licenses are required for each tool used.
- **Verify Component** – For each DesignWare component, tests and component-specific testbenches are provided to demonstrate the functionality of the component. The tests and testbenches used in these activities are not reusable, however, the simulation waveforms are useful to examine detailed signal behavior.
- **Verify Subsystem** – The Formal Verification activity runs Formality (license required) on the entire synthesized subsystem. The Simulate Subsystem activity generates a custom testbench for your specific subsystem configuration and executes basic connectivity tests (also called “ping” tests). The testbench and tests created by this activity can be reused as a starter testbench for a larger subsystem or more detailed tests.
- **Create Component GTECH Simulation Model** – If VCS is the simulator used for the Subsystem Simulation or Verify Component activities, this activity group is not required. However, if a simulator other than VCS is selected, and you do not own a source license for the RTL of the DesignWare component, a GTECH simulation model is required and these activities must be completed before simulation can be run.
- **Export Subsystem** – If you intend to package the subsystem for re-use in different designs or to package your own subsystem IP for delivery to your customers, these activities can be used. They require a coreBuilder license. For more information, see the *coreBuilder User Guide*.

Integration Considerations

When integrating multiple DesignWare IP components, much of the detail is handled automatically by coreAssembler:

- Component inputs/outputs that belong to standard interfaces (for example, the AXI protocol) are connected automatically.
- Component inputs/outputs that do not belong to standard interfaces (for example, interrupt pins) are automatically handled with default connections that are pre-defined by the DesignWare IP designer.
- Interface parameters (for example, AXI address bus width or AHB data bus width) are defined one time, and then propagated to connected components automatically.
- Configured RTL generation for both components and the top level of the subsystem occurs automatically.
- Subsystem-level synthesis and verification are highly automated.

Performance/Area Trade-off

Depending on system requirements, you can configure and connect DMA controller(s) to the ports of DW_axi via DW_axi_hmx modules with a trade-off between area and performance.

Let's take an example where a system requires four parallel DMA operations; that is, four parallel DMA channels are required.

Highest Performance

Configuring the subsystem to achieve the highest performance requires the following components:

- Two, two-channel four AHB master interface DMA Controllers
- Eight DW_axi_hmx interface modules
- Eight DW_axi ports

Figure 12 illustrates a subsystem configuration that is configured for the highest performance (highest area).

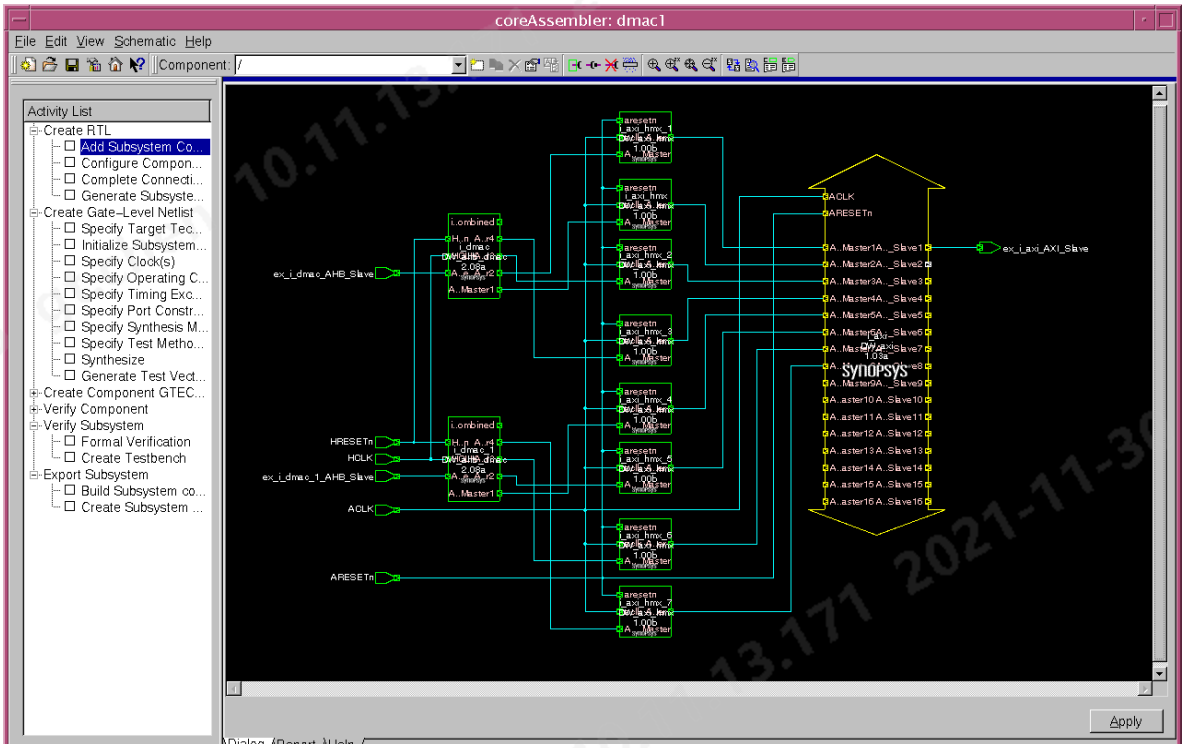


Figure 12: Subsystem Configuration for Highest Performance (Highest Area)

The setup for the highest performance configuration is as follows:

1. Configure two DMA Controllers – Each DMA Controller has two channels and four master interfaces.
   a. There are four DMA channels: DMA1_CH1, DMA1_CH2, DMA2_CH1 and DMA2_CH2, where DMA1 is the first DMA controller and DMA2 is the second DMA controller.
   b. There is a total of eight DMA master interfaces, four on each DMA, DMA1_M1, DMA1_M2, DMA1_M3, DMA1_M4 on first DMA controller, DMA2_M1, DMA2_M2, DMA2_M3 and DMA2_M4 on the second DMA controller.

Configure the DMA channels and master interfaces as follows:

Source:	DMA1_CH1 <-> DMA1_M1
Destination	DMA1_CH1 <-> DMA1_M2
Source	DMA1_CH2 <-> DMA1_M3
Destination	DMA1_CH2 <-> DMA1_M4
Source	DMA2_CH1 <-> DMA2_M1
Destination	DMA2_CH1 <-> DMA2_M2
Source	DMA2_CH2 <-> DMA2_M3
Destination	DMA2_CH2 <-> DMA2_M4

2. All eight DMA master interfaces have a DW_axi_hmx in front of them.

   This setup allows:

   - Four outstanding read commands
   - Four outstanding write commands
   - Read data and write data interleaving between channels
   - Separate read and write command channels

Lowest Area

Configuring the subsystem for lowest area requires the following components:

- Single, four-channel four-AHB-master interface DMA Controller
- Four DW_axi_hmx interface modules
- Four DW_axi ports

Figure 12 illustrates a subsystem configuration that is configured for the lowest area (lowest performance).

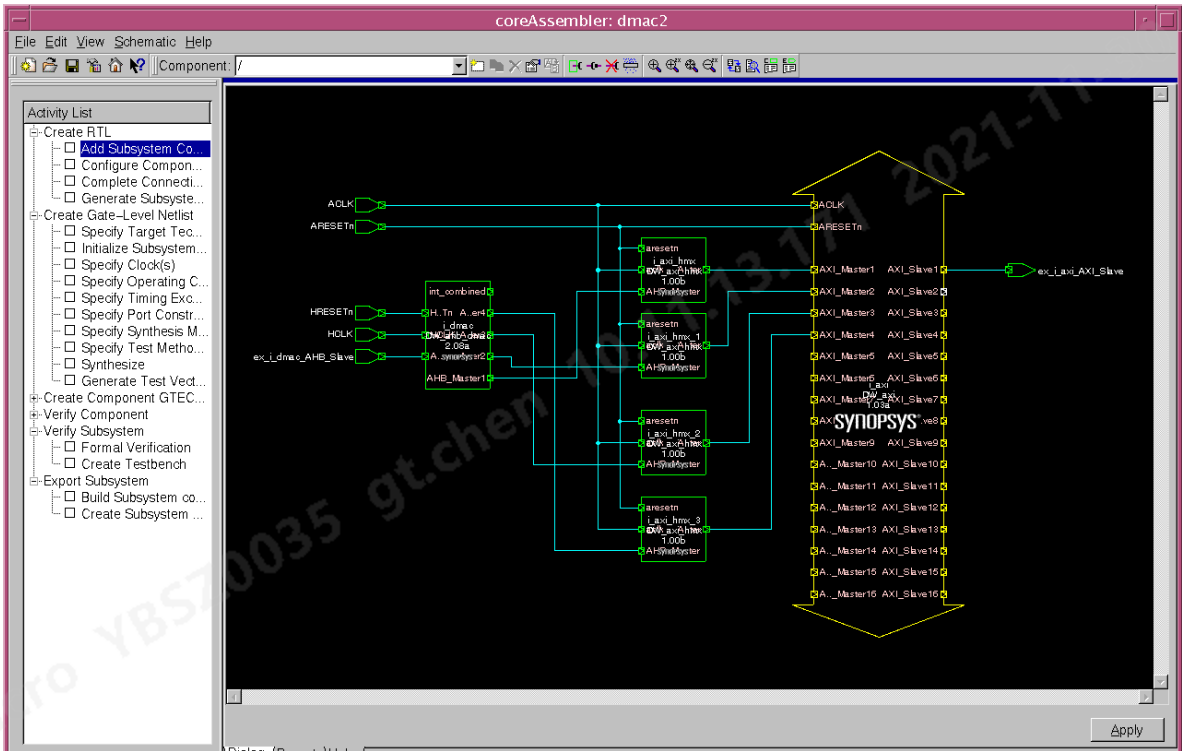


Figure 13: Subsystem Configuration for Lowest Area (Lowest Performance)

The setup for the lowest area configuration is as follows:

1. Configure a single DMA Controller with four channels and four master interfaces:
   a. The four DMA channels are: DMA1_CH1, DMA1_CH2, DMA1_CH3 and DMA1_CH4.
   b. There are four DMA master interfaces: DMA1_M1, DMA1_M2, DMA1_M3 and DMA1_M4.

   Configure the DMA channels and interfaces as follows:

Source	DMA1_CH1 <-> DMA1_M1
Destination	DMA1_CH1 <-> DMA1_M2
Source	DMA1_CH2 <-> DMA1_M1
Destination	DMA1_CH2 <-> DMA1_M2
Source	DMA2_CH3 <-> DMA1_M3
Destination	DMA2_CH3 <-> DMA1_M4
Source	DMA2_CH4 <-> DMA1_M3
Destination	DMA2_CH4 <-> DMA1_M4

2. All four DMA master interfaces have an HMX in front of them.

   This setup allows:

   - Separation of read and write command and data channels.
   - A possible read outstanding command depth of 2 if the read commands originate from different master interfaces, i.e. there can only be a single read active command between channels 3 and 4. However, there could be, for example, an active read command from DMA1_CH1 or DMA1_CH2, and DMA1_CH3 or DMA1_CH4.
   - Possible read/data interleaving with the same restrictions as described above.

Defined-Length Burst Support on DMAC

By default, the DW_ahb_dmac supports only incremental (INCR) bursts. To achieve better performance, defined length bursts—such as INCR4, INCR8 and INCR16—are required.

The DW_ahb_dmac can be configured to use defined-length bursts by setting the configuration parameter DMAH_INCR_BURSTS to 0. In this mode, the DW_ahb_dmac selects the largest valid defined-length burst to complete the transfer.

Reference Documentation

The following documents can be found in the *Guide to DesignWare AMBA IP Components Documentation*. Upon installation of the DesignWare Synthesizable IP for AMBA, this document can also be found on your local system at $DESIGNWARE_HOME/doc/amba/latest/intro.pdf.

- DW_axi_hmx Databook and Release Notes – For detailed information on how commands are transferred from an AMBA 2.0 AHB master to AMBA 3 AXI Interconnect Fabric, refer to Chapter 3 (Functional Description) and Chapter 7 (Integration Considerations) in the DW_axi_hmx databook.
- DW_axi_x2h Databook and Release Notes – For detailed information on how commands are transferred from AXI to AHB, refer to Chapter 3 (Functional Description) and Chapter 7 (Integration Considerations) in the DW_axi_x2h databook.
- DW_axi Databook and Release Notes – Provides AXI interconnect details.

- DW_ahb Databook and Release Notes – Provides AHB interconnect details.
- DW_ahb_dmac Databook and Release Notes – Provides AHB DMA Controller details.
- Using coreAssembler with DesignWare Synthesizable IP – Chapter 6 provides tutorials for getting started with coreAssembler and DesignWare IP.
- *coreAssembler User Guide* – Provides general coreAssembler usage.
- AMBA 2.0 Specification – *Download here*.
- AMBA 3 Specification – *Download here*.