Introduction

For years, system designers have been manually connecting IP peripheral functions to embedded processors, taking anywhere from weeks to months to accomplish. With the advent of SOPC Builder from Altera®, designers can now define, parameterize, and link intellectual property (IP) cores automatically to create a custom system-on-a-programmable-chip in minutes. Altera provides two highly complex and efficient processors: the soft core Nios® embedded processor, and Excalibur™ devices, with the hard core ARM922T®. Some applications need more processing power than one microprocessor can provide. One alternative to higher performance processing is to run more than one processor in the system. For example, multiple Nios processors can be instantiated in the FPGA portion of an Excalibur device. SOPC Builder supports the following two bus standards for interconnecting these processors and peripherals:

- The advanced micro-controller bus architecture (AMBA) high-performance bus (AHB) for Excalibur devices
- The Avalon™ bus for the Nios embedded processor

SOPC Builder provides two bridges to interface modules that reside on an AHB with modules on an Avalon bus. The AHB-to-Avalon bridge interfaces an AHB master with an Avalon slave; the Avalon-to-AHB bridge interfaces an Avalon master with an AHB slave. Both the Avalon and AHB specifications use slave-side arbitration (referred to as multi-layer AHB in the AHB specification), which places an arbiter in front of every slave that has multiple masters connected to it. This document describes the features of these bridges and the bus transactions across the bridges.

AHB Signal Specifications

The AHB is designed for high-performance, high clock frequency system modules. The AHB provides an efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral functions. Table 1 shows the definition of each of the AHB signals in the bridges.

For more information on the AHB, refer to the [AMBA Specification Revision 2.0](#).

<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hclk</td>
<td>Clock source</td>
<td>Bus clock. This clock times all bus transfers. All signal timings are related to the rising edge of HCLK.</td>
</tr>
<tr>
<td>hresetn</td>
<td>Reset controller</td>
<td>Reset. The bus reset signal is active low and is used to reset the system and the bus. This signal is the only active low signal.</td>
</tr>
<tr>
<td>haddr[31:0]</td>
<td>Master</td>
<td>Address bus. The 32-bit system address bus.</td>
</tr>
<tr>
<td>htrans[1:0]</td>
<td>Master</td>
<td>Transfer type. Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.</td>
</tr>
<tr>
<td>hwrite</td>
<td>Master</td>
<td>Transfer direction. When high this signal indicates a write transfer; when low a read transfer.</td>
</tr>
<tr>
<td>hsize[2:0]</td>
<td>Master</td>
<td>Transfer size. Indicates the size of the transfer, which is typically byte (8 bit), halfword (16 bit) or word (32 bit).</td>
</tr>
<tr>
<td>hburst[2:0]</td>
<td>Master</td>
<td>Burst type. Indicates if the transfer forms part of a burst. Four, eight and sixteen beat bursts are supported and the burst may be either incrementing or wrapping.</td>
</tr>
<tr>
<td>hwdata[31:0]</td>
<td>Master</td>
<td>Write data bus. The write data bus is used to transfer data from the master to the bus slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.</td>
</tr>
<tr>
<td>hselx</td>
<td>Decoder</td>
<td>Slave select. Each AHB slave has its own slave select signal and this signal indicates that the current transfer is intended for the selected slave. This signal is simply a combinatorial decode of the address bus.</td>
</tr>
<tr>
<td>hrdata[31:0]</td>
<td>Slave</td>
<td>Read data bus. The read data bus is used to transfer data from bus slaves to the bus master during read operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.</td>
</tr>
</tbody>
</table>
Notes:
(1) A 3-bit signal in the AHB specification. However, the AHB-to-Avalon and Avalon-to-AHB bridges only use the lower two bits, because the maximum bus width is 32 bits for the bridges.
(2) The hselx signal is not included in the Avalon-to-AHB bridge, because the bridge passes all transactions through to the AHB side. Therefore, address decoding must be done outside the bridge.
(3) The hbusreqx and hgrantx signals are not included in the AHB-to-Avalon bridge, because these signals are between the AHB master and the AHB arbiter.

Avalon Signal Specifications

The Avalon bus is a simple bus for connecting microprocessors and peripherals into a system-on-a-programmable-chip. The Avalon bus has a low logic element (LE) count for bus logic and fully synchronous operation.

Table 2 shows the Avalon bus signals, some of which are included on the bridges.

For more information on the Avalon bus, refer to the Avalon Bus Specification.

Table 2. Avalon Bus Signals

<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>Clock</td>
<td>The system-wide clock, which is provided to all Avalon devices.</td>
</tr>
<tr>
<td>reset_n</td>
<td>System</td>
<td>The system-wide reset, which is provided to all Avalon devices.</td>
</tr>
<tr>
<td>address</td>
<td>Master</td>
<td>The address bus. The Avalon master drives the address bus, which is broadcast (shared) by all slave modules.</td>
</tr>
<tr>
<td>writedata</td>
<td>Master</td>
<td>The write data bus. The Avalon master drives an outgoing (write) data bus, which is broadcast (shared) by all slave modules.</td>
</tr>
<tr>
<td>readdata</td>
<td>Slave</td>
<td>The read data bus. Each slave device in an Avalon system produces its own unique (unshared) read-data output.</td>
</tr>
<tr>
<td>chipselect</td>
<td>Master</td>
<td>The device-select signal. Individual select-signals are decoded for each slave-module in the Avalon system. A slave module should ignore all of its other Avalon bus inputs unless its chipselect-type port is driven logic true.</td>
</tr>
<tr>
<td>byteenable</td>
<td>Master</td>
<td>The byte-enable bus. Some slave devices may support individual, byte-wide read and write operations. Such devices use byteenable to select which bytes are to be used for the current bus transaction.</td>
</tr>
<tr>
<td>write</td>
<td>Master</td>
<td>The write strobe. The Avalon master drives an outgoing write-strobe control signal, which is broadcast to (shared by) all slave modules. The master asserts this signal (drives it high) to indicate that the current bus transaction is a write-data operation to the selected slave.</td>
</tr>
<tr>
<td>read</td>
<td>Master</td>
<td>The read strobe. The Avalon master drives an outgoing read-strobe control signal, which is broadcast to (shared by) all slave modules. The master asserts this signal (drives it high) to indicate that the current bus transaction is a read-data operation to the selected slave.</td>
</tr>
<tr>
<td>waitrequest</td>
<td>Slave</td>
<td>The interrupt request. If a slave drives a logic true signal on its waitrequest port when it is selected, the current Avalon bus transaction is extended until the waitrequest line is deasserted.</td>
</tr>
</tbody>
</table>

Notes:
(1) The chipselect signal is not a part of the AHB-to-Avalon bridge, because the Avalon bus module generates the chipselect by decoding the address.
AHB-to-Avalon Bridge

This AHB-to-Avalon bridge is an interface between an AHB master and an Avalon slave. Figure 1 shows the interface signals of the AHB-to-Avalon bridge.

Figure 1. AHB-to-Avalon Bridge Interface Signals

For more information on AHB signals and AHB transactions, refer to AMBA Specification revision 2.0. For more information on the Avalon bus, please refer to the Avalon Bus Specification.

The AHB-to-Avalon bridge supports all AMBA transactions types by the AHB master i.e., SINGLE, INCR, INCR4, INCR8, INCR16, WRAP4, WRAP8 and WRAP16. The Verilog HDL source code for this bridge is provided with this paper. All AHB transactions are converted to a fundamental Avalon transfer (non-streaming, without latency). For more information on fundamental Avalon transfers, refer to the Avalon Bus Specification.

AHB-to-Avalon Bridge Features

Whenever an AHB master starts a data transaction, the bridge always responds with an OKAY (hresp = 00) response. Hence, you must know the sequence of events of your design and create an AHB master accordingly. The AHB-to-Avalon bridge has the following features:

- The AHB-to-Avalon bridge converts all AHB transactions to single Avalon transactions with wait-states (when necessary).
- For every transaction, the AHB-to-Avalon bridge responds to the AHB master with OKAY (hresp[1:0] = 00).
- The AHB-to-Avalon bridge supports one AHB master and multiple Avalon slaves. Multiple bridges can be used in the case where multiple AHB masters are connected to the Avalon bus.
- The same clock is used for the whole bridge. Therefore, if the Avalon slave is in a different clock domain than the AHB master, user logic must be added to synchronize the data across the clock domains.
- The AHB master does not need to take into account the setup or hold cycle requirements of the Avalon slaves. This is taken care of by the Avalon bus module.
Transfers with latency from an Avalon slave are translated into transactions with wait-states before going to the AHB-to-Avalon bridge. This is true for any transaction on the Avalon bus in which the master is not latency aware.

The waitrequest signal from the Avalon slave is inverted and tied to the hreadyo signal of the AHB master. When the waitrequest signal is high (and hence the hreadyo signal is low), the current transaction from the AHB master is suspended until the Avalon slave drives the waitrequest signal low.

The hsize control signal and haddr[1:0] from the AHB master are used to determine the alignment of data between the AHB master and the Avalon slave using byteenable[3:0] on the Avalon bus.

**AHB-to-Avalon Bridge Transactions**

Figure 2 shows the AHB master on the AHB-to-Avalon bridge connected to multiple slaves that sit on the Avalon bus. All of these Avalon slaves should be within the address range of the AHB master.

**Figure 2. AHB-to-Avalon Bridge Master Slave Connections**

For more information on designing an AHB master, refer to the *AMBA Specification Revision 2.0*.

The following steps detail a transaction from the AHB master to a slave on the Avalon bus.

1. The AHB master starts a transaction by driving haddr[31:0] and the AHB control signals.

2. These AHB signals are driven into the AHB-to-Avalon bridge, converted to Avalon signals, and driven onto the Avalon bus. The AHB specification has separate address and data phases while the Avalon specification requires the address and data to be presented during the same cycle. Therefore, the AHB-to-Avalon bridge delays the address, to align the address and data on the Avalon bus.

3. The Avalon bus module decodes the address and asserts chipselect for the appropriate Avalon slave. On the rising edge of the clock, the Avalon slave samples chipselect, address[31:0], and other Avalon control signals.
4. The Avalon slave may set the `waitrequest` signal high, which indicates that it cannot complete the transaction by the next rising clock edge.

5. At the next rising edge of the clock, the AHB master receives the slave’s wait request via the `hreadyo` signal which is logically tied to the inverse of the `waitrequest` signal. By default, an AHB address phase lasts for one clock cycle. The AHB master extends its address phase, if the slave is not ready.

6. When the Avalon slave has finished processing the data, it clears `waitrequest` and completes the transaction. For example, if it is a read transaction, the Avalon slave sends data on `readdata`. According to the Avalon specification for a write transaction, the Avalon slave does not send any signal back to the bridge to indicate the end of the transaction.

**Avalon-to-AHB Bridge**

The Avalon-to-AHB bridge serves as an interface between an Avalon master and an AHB slave. Figure 3 shows the signals of the Avalon-to-AHB bridge.

For more information on AHB signals and AHB transactions, refer to *AMBA Specification version 2.0* and above. For more information on the Avalon bus specification, refer to the *Avalon Bus Specification*.

The Avalon-to-AHB bridge converts all Avalon transactions, including streaming transfers and transfers with latency, into AHB non-sequential single transfers. The Verilog HDL source code for this bridge is provided with this paper.

*Figure 3. Avalon-to-AHB Bridge (1), (2), (3)*

**Note:**

(1) The `hsize[2:0]` signal is a 3 bit-signal in the AHB specification. However, the AHB-to-Avalon and Avalon-to-AHB bridges only use the lower two bits, because the maximum bus width is 32 bits for the bridges.

(2) The bridge does not support address decoding, so you must generate `h sel` outside of the bridge.

(3) This bridge does not support `hreadyi`. The slave uses `hready` to insert wait states.

**Avalon-to-AHB Bridge Features**

When the Avalon master starts a data transaction, the Avalon-to-AHB bridge converts this transaction into non-sequential single transfers to the AHB slave. The Avalon-to-AHB bridge has the following features:
- Converts all Avalon transactions to AHB non-sequential single transfers (htrans[1:0] = 10 and hburst[2:0] = 000) followed by an IDLE cycle (htrans[1:0] = 00). This IDLE cycle adds one cycle to every transaction, so there is at least one wait state on the Avalon bus. Therefore, waitrequest is high for the first cycle of every transaction.
- The same clock is used for the whole bridge. Therefore, if the Avalon master is in a different clock domain than the AHB slave, user logic must be added to synchronize the data across the clock domains.
- The waitrequest output signal from the bridge suspends the current transaction until the AHB slave is ready to complete the transaction. The waitrequest signal is tied to the combination of AHB signals (including hready, hgrant, hbusreq).
- The bridge sets hsize[1:0] and haddr[1:0] based on byteenable[3:0], which is provided by the Avalon master.

**Avalon-to-AHB Bridge Transactions**

Figure 4 shows the Avalon master on the Avalon-to-AHB bridge connected to multiple slaves that sit on the AHB. All of these AHB slaves should be within the address range of the Avalon master. Multiple masters can be connected to the Avalon-to-AHB bridge. SOPC Builder takes care of the arbitration.

For more information on simultaneous multi-mastering, refer to *AN 184: Simultaneous Multi-Mastering with the Avalon Bus*.

**Figure 4. Avalon-to-AHB Bridge Master Slave Connections**

For more information on designing an AHB master, refer to the *AMBA Specification Revision 2.0*.

The following steps detail a transaction from the Avalon master to the AHB slave:
1. The Avalon-to-AHB bridge starts data transactions only when it is granted the AHB by the AHB arbiter via the hgrant signal. The bridge requests control of the AHB via the hbusreq signal.

2. The Avalon master starts a transaction by driving address[31:0] along with the Avalon control signals.

3. The bridge translates the Avalon signals into AHB signals and sends them to the AHB.

4. The AHB slave samples haddr[31:0] and other AHB control signals on the rising edge of the clock.

5. The AHB slave may set the hready signal low indicating that the slave is inserting wait states.

6. At the next rising edge of the clock, the Avalon master receives the slave's wait state via the waitrequest signal (which is tied to the combination of the hready, hgrant and hbusreq).

7. When the AHB slave has finished processing the data, it asserts hready and completes the transaction. For a read transaction, the AHB slave presents data on hrdata[31:0] and the Avalon master samples this data on readdata[31:0].