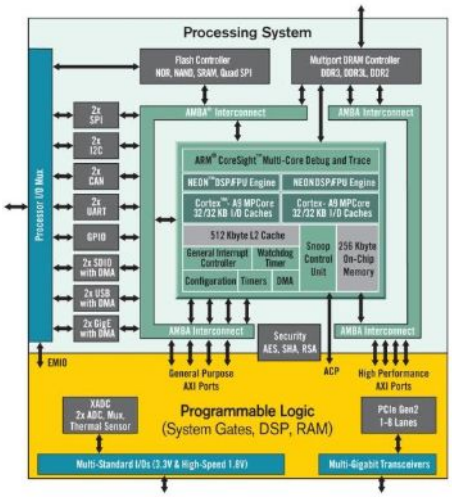
Designing a Custom AXI-lite Slave Peripheral

Version 1.0, July 2014
Rich Griffin (Xilinx Embedded Specialist, Silica EMEA)

Overview



Xilinx Zynq-7000 All Programmable SoC devices offer architectural features which offer designers an advanced ARM based processing system, coupled with a flexible region of Programmable Logic. The Xilinx Vivado tool suite offers the designer a powerful and flexible design environment, including support for designing and testing custom IP blocks which can be used, and re-used, in multiple designs.

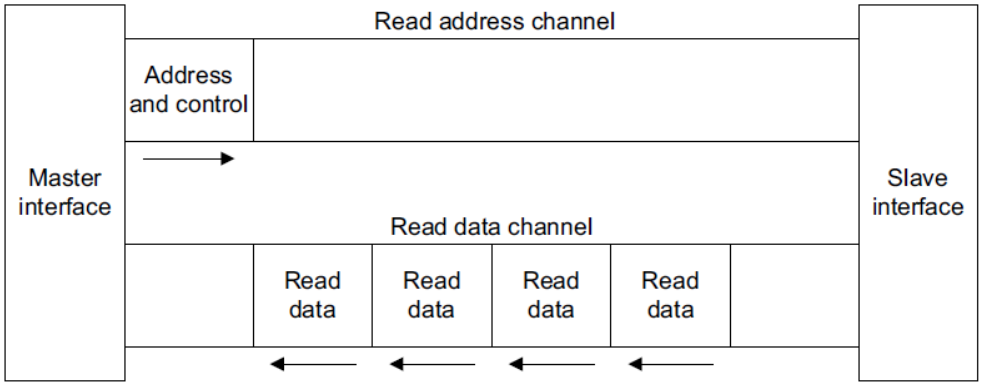
The Processing System (PS) includes two Cortex-A9 processor cores, a dedicated DDR memory controller, and a wide selection of IO peripherals. The Programmable Logic (PL) is based on the Xilinx 7-Series FPGA fabric and offers the designer the ability to implement their own custom logic which can work alongside the software running on the processor cores. The two areas of the device, PS and PL, are linked by a series of interfaces which adhere to the AXI4 interconnect standard. These interfaces allow the designer to implement custom logic in the PL which can be connected to the PS and extend the range of peripherals which are available and visible in the processor's memory map. One of the most common uses of this technology is to create a block of custom logic in the PL, and then add control and status monitoring capabilities by using memory mapped registers which the processors can access via the AXI4 interconnect. The high performance of dedicated custom logic can then be utilised in the system, without sacrificing the flexibility of software for control and status monitoring tasks.

The AXI4 (Advanced eXtensible Interface) protocol has been designed to offer different variants of the interconnect. The full AXI4 specification offers a diverse range of powerful features including variable data and address bus widths, high bandwidth burst operations, advanced caching support, out of order transaction completion, and various transaction protection and access permissions. Although this specification provides the user with enormous flexibility and control, it is often desirable to implement a much simpler peripheral which uses only a subset of these features. For that reason, a reduced feature variant of the AXI4 specification exists in the form of "AXI4-lite". This subset of the specification supports uses cases where only basic interconnect transactions are required, and removes some of the more advanced capabilities of the interconnect such as cache support, burst support, and variable bit widths for the address and data buses. The AX4-lite interconnect is therefore perfect for applications where simple control and status monitoring capabilities are required for a custom built IP block. This guide discusses how to build a custom IP block in the PL, implement memory mapped registers, and make them available via the AXI4-lite interconnect so that they are visible to the processor. This guide will also discuss the creation of some basic device drivers, showing how software can be written to access the registers on the custom peripheral.

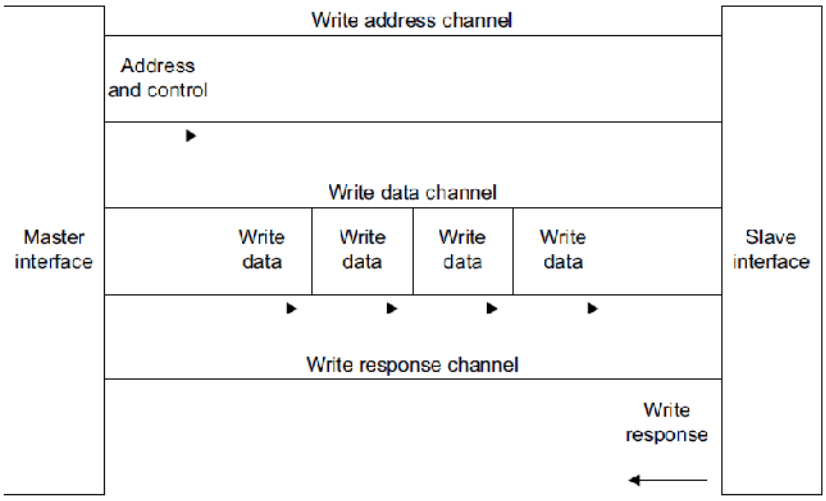
AXI4-lite Protocol; The Five Channels

The AXI4-lite protocol has been written with a modular design flow in mind. There are five channels which make up the specification; the read address channel, the write address channel, the read data channel, the write data channel, and the write acknowledge channel. It is important to understand the function of each of these channels before starting to develop the custom IP block. For the purposes of clarity, this guide discusses the concepts of read and write transactions from the processor's perspective.

The read address channel allows the processor to signal that it wishes to initiate a read transaction. As the name suggests, this channel carries addressing information and some handshaking signals. The read data channel carries the data values that are transferred during a transaction, along with their associated handshaking signals. Together, these two channels contain everything that is needed for a successful read transaction.



The write address channel allows the processor to signal that it wishes to initiate a write transaction. It is identical to the read address channel in every other way. The write data channel performs an equivalent function to be read data channel, except that the data flows in the opposite direction (i.e. towards the slave). For write transactions an additional channel is used in the form of the Write Response Channel. This last channel is used to allow the slave peripheral to acknowledge receipt of the data, or to signal that an error has occurred.



The next important aspect of AXI4-lite that must be understood is the handshaking signals. These signals are consistent across the five channels, and offer the user a simple yet powerful way to control all read and write transactions. The signals are based on a simple "Ready" and "Valid" principle; "Ready" is used by the recipient to indicate that it is ready to accept a transfer of a data or address value, and "Valid" is used to clarify that the data (or address) provided on that channel by the sender is valid so that the recipient can then sample it.

Taking the example of the Write Address Channel, the following signals are used:

AXI4-lite Read Address Channel			
Signal Name	Size	Driven by	Description
S_AXI_ARADDR	32 bits	Master	Address bus from AXI interconnect to slave peripheral.
S_AXI_ARVALID	1 bit	Master	Valid signal, asserting that the S_AXI_AWADDR can be sampled by the slave peripheral.
S_AXI_ARREADY	1 bit	Slave	Ready signal, indicating that the slave is ready to accept the value on S_AXI_AWADDR.

It is important to understand that a "Valid" signal can be asserted before the recipient is ready to receive it, but when this scheme is used the value presented by the sender must be held in the "Valid" state until the receiver is ready to proceed with the transaction. It is equally acceptable for the receiver to indicate a "Ready" state before the sender makes a value "Valid". In this latter case, the sender can be assured that the transfer will complete within one clock cycle, because it has forewarning of the receiver's readiness.

A frequently misunderstood use of the Valid and Ready signals, and one which often results in incorrect and illegal implementations of the AXI4-lite protocol, is the assumption that the sender can/must wait for "Ready" to be asserted by the receiver before it asserts its "Valid" signal. This is an illegal use of the handshaking signals and can result in a deadlock situation arising. Ready can be asserted before Valid, but the sender must never wait for Ready as a pre-condition to commencing the transaction.

This important aspect of the AXI4-lite protocol can be easily remembered by applying the "Assert and wait" rule. Never use the "Wait before Assert" approach, because this is illegal.

"Assert Ready and wait for Valid" ✓

"Assert Valid and wait for Ready" ✓

"Wait for Ready before asserting Valid" ✗

This is an important learning point. Make sure that this concept of handshaking is clearly understood before continuing.

The use of the Ready / Valid handshaking scheme is identical across all five AXI4-lite channels.

Read Transactions

Starting with the example of a read transaction, the signals belonging to the Read Data channel can now be examined.

AXI4-lite Read Data Channel			
Signal Name	Size	Driven by	Description
S_AXI_RDATA	32 bits	Slave	Data bus from the slave peripheral to the AXI interconnect.
S_AXI_RVALID	1 bit	Slave	Valid signal, asserting that the S_AXI_RDATA can be sampled by the Master.
S_AXI_RREADY	1 bit	Master	Ready signal, indicating that the Master is ready to accept the value on the other signals.
S_AXI_RRESP	2 bits	Slave	A "Response" status signal showing whether the transaction completed successfully or whether there was an error.

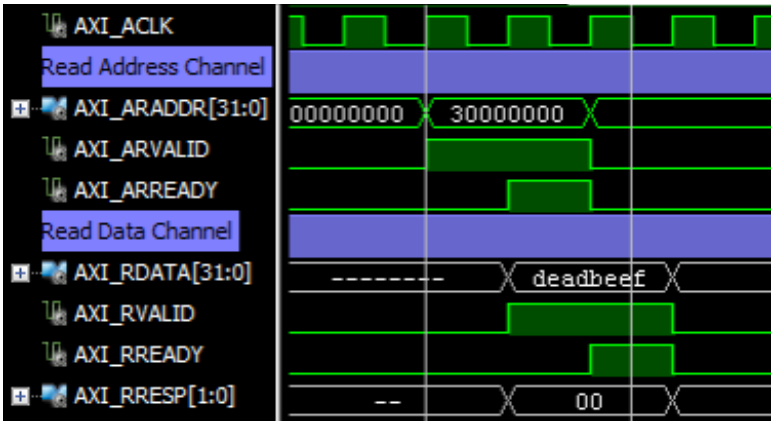
The transaction in this example is one of the Master (e.g. the processor) reading a data value from the Slave (e.g. the peripheral). The important difference to recognise here is that the roles of driving Valid and Ready have now been reversed. For the purposes of data flow, the sender is now the Slave and the receiver is now the Master. Therefore, the Slave should provide the data value on S_AXI_RDATA and assert S_AXI_RVALID to qualify it. A new signal is introduced at this stage, S_AXI_RRESP, which is a status signal that describes whether or not the transaction has completed successfully. The Master can then assert S_AXI_RREADY to indicate that it is ready to accept the data (or, optionally, the Master could assert S_AXI_RREADY prior to the assertion of S_AXI_RVALID, indicating that it is ready to receive the data as soon as S_AXI_RVALID is asserted by the slave).

The coding of the S_AXI_RRESP signal should be discussed at this point, so as to understand its purpose. In most situations the RRESP signal will be tied to "00" because the more advanced features will not be needed. However, some of the more advanced functionality of the RRESP signals may occasionally be useful, usually when then design of the peripheral needs to signal a "retry" condition due to a delay in the availability of the data.

AXI4-lite Response Signalling		
RRESP State [1:0]	Condition	Description
00	OKAY	**"OKAY"** The data was received successfully, and there were no errors.
01	EXOKAY	**"Exclusive Access OK"** This state is only used in the full implementation of AXI4, and therefore cannot occur when using AXI4-Lite.
10	SLVERR	**"Slave Error"** The slave has received the address phase of the transaction correctly, but needs to signal an error condition to the master. This often results in a retry condition occurring.
11	DECERR	**"Decode Error"** This condition is not normally asserted by a peripheral, but can be asserted by the AXI interconnect logic which sits between the slave and the master. This condition is usually used to indicate that the address provided doesn't exist in the address space of the AXI interconnect.

The timing diagram for a complete read transaction, showing the use of the Read Address and Read Data channels, is shown below. In this example the Valid signals are asserted before the Ready signals on both channels, and it can be clearly seen that the transaction on each channel completes one clock cycle after Ready is asserted. If the Ready signals were

pre-emptively asserted by the receiver in each case, the entire transaction could feasibly be shortened by two clock cycles, leaving the Valid signals asserted for just one clock cycle each. The use of the "00" AXI_RRESP state can also be clearly seen, indicating that the transaction completed successfully.



Write Transactions

Write transactions are almost identical to the Read transactions discussed above, except that the Write Data Channel has one signal that is different to the Read Data Channel. The S_AXI_WSTRB (Write STRoBe) controls which of the bytes in the data bus are valid and should be read by the Slave. For example, if the Master is performing an 8 bit write transaction to the slave then not all 32 bits of the data bus will be relevant. In the case of an AXI4-lite transaction the data bus is 32 bits wide, and therefore the WSTRB signal is four bits wide with each bit representing one byte (8 bits) of data.

AXI4-lite Write Data Channel			
Signal Name	Size	Driven by	Description
S_AXI_WDATA	32 bits	Master	Data bus from the Master / AXI interconnect to the Slave peripheral.
S_AXI_WVALID	1 bit	Master	Valid signal, asserting that the S_AXI_RDATA can be sampled by the Master.
S_AXI_WREADY	1 bit	Slave	Ready signal, indicating that the Master is ready to accept the value on the other signals.
S_AXI_WSTRB	4 bits	Master	A "Strobe" status signal showing which bytes of the data bus are valid and should be read by the Slave.

The write strobe signals can often be a source of confusion, specifically around which strobe signals relate to which bits of the data bus. The AXI specification describes this using an accurate and unequivocal formula "WSTRB[n] corresponds to WDATA[(8 × n) + 7:(8 × n)]", but sadly this is not always easy to comprehend at first glance. The following table serves to illustrate some practical examples of strobe usage. For the purposes of these (non exhaustive) examples, the "active" bits are shown as logic 1, and the "inactive" bits are shown as logic 0.

S_AXI_WSTRB signals		
S_AXI_WSTRB [3:0]	S_AXI_WDATA active bits [31:0]	Description
1111	11111111111111111111111111111111	All bits active
0011	00000000000000001111111111111111	Least significant 16 bits active
0001	00000000000000000000000011111111	Least significant byte (8 bits) active.
1100	11111111111111110000000000000000	Most significant 16 bits active

During a write transaction, the last of the five AXI channels (The Write Response Channel) is also used because the Slave needs a way to signal to the Master that the data was received correctly. The same Ready / Valid handshaking style is used as in the other channels. A simple 2 bit response system is used to signal a successful transaction or an error, coded in the same way as with the other RESP signals (e.g. "00" for a successful transaction, and similar codings for the error status conditions).

AXI4-lite Write Response Channel			
Signal Name	Size	Driven by	Description
S_AXI_BREADY	1 bit	Master	Ready signal, indicating that the Master is ready to accept the "BRESP" response signal from the slave.
S_AXI_BRESP	2 bits	Slave	A "Response" status signal showing whether the transaction completed successfully or whether there was an error.
S_AXI_BVALID	1 bit	Slave	Valid signal, asserting that the S_AXI_BRESP can be sampled by the Master.

A timing diagram of a complete AXI4-lite write transaction is shown below, showing the use of the three AXI write channels. In this example the value of 0x00000008 is being written to address 0x30000000. Note that this is a 32 bit write, and all four write strobe (WSTRB) signals are asserted. The use of the handshaking signals on all three channels is shown below, using the same "valid before ready" scheme on the address and data channels, and a "ready before valid" scheme on the write response channel. The two different handshaking schemes are not used here for any specific reason, other than to demonstrate an example of each style. Both are perfectly acceptable, and can be used in combination across the five AXI4-lite channels.