Note sul protocollo di bus AXI

Da wikipedia (https://en.wikipedia.org/wiki/Advanced_eXtensible_Interface)

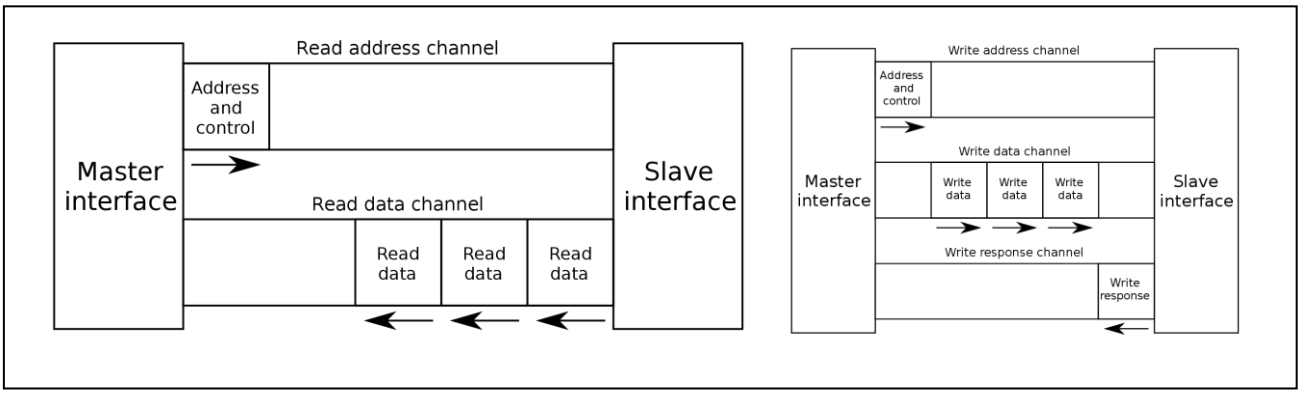
"The **Advanced eXtensible Interface** (**AXI**) is an on-chip communication bus protocol and is part of the Advanced Microcontroller Bus Architecture specification (AMBA). AXI had been introduced in 2003 with the AMBA3 specification. In 2010, a new revision of AMBA, AMBA4, defined the AXI4, AXI4-Lite and AXI4-Stream protocols. AXI is royalty-free and its specification is freely available from ARM."

La comunicazione fra IP è di tipo punto-punto. Di due IP collegati mediante AXI, uno svolge le funzioni di Master e l'altro le funzioni di Slave. Sono previsti cinque canali di comunicazione con handshake indipendenti. Tre canali sono usati per la scrittura dal Master verso lo Slave; i rimanenti due canali sono usati per le operazioni di lettura dal Master verso lo Slave.

I cinque canali sono:

- Read Address channel (AR)
- Read Data channel (R)
- Write Address channel (AW)
- Write Data channel (W)
- Write Response channel (B)

I canali coinvolti nelle operazioni di lettura e di scrittura e la sequenza temporale delle operazioni possono essere schematizzati come nella figura seguente (fonte wikipedia):



Il protocollo AXI è di tipo "burst based", il che vuol dire che una volta fornito un indirizzo, l'operazione di lettura/scrittura può coinvolgere più di un singolo dato.

Lo AXI-Lite è un protocollo semplificato in cui (fonte wikipedia):

AXI4-Lite is a subset of the AXI4 protocol, providing a register-like structure with reduced features and complexity. Notable differences are:

- all bursts are composed by 1 beat only
- all data accesses use the full data bus width, which can be either 32 or 64 bits

AXI4-Lite removes part of the AXI4 signals but follows the AXI4 specification for the remaining ones. Being a subset of AXI4, AXI4-Lite transactions are fully compatible with AXI4 devices, permitting the interoperability between AXI4-Lite initiators and AXI4 targets without additional conversion logic.[16]

Il numero di segnali e la complessità del protocollo AXI-Lite risultano considerevolmente ridotti.

Oltre al segnale di clock (ACLK) e al reset (ARESETN), sono previsti 19 segnali distinti (Wikipedia):

Write address channel	Write data channel	Write response channel	Read address channel	Read data channel
AWVALID	WVALID	BVALID	ARVALID	RVALID
AWREADY	WREADY	BREADY	ARREADY	RREADY
AWADDR	WDATA	BRESP	ARADDR	RDATA
AWPROT	WSTRB		ARPROT	RRESP

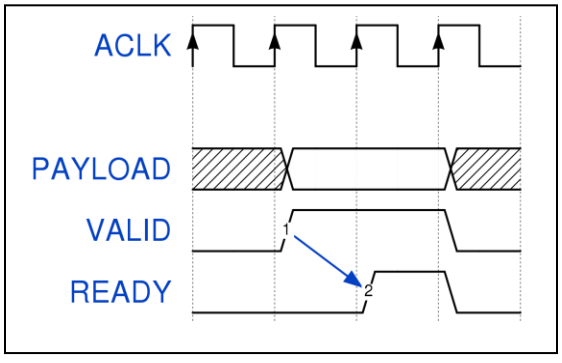
Sempre da wikipedia:

Other than some basic ordering rules, each channel is independent from each other and has its own couple of `xVALID/xREADY` handshake signals.

Il meccanismo di handshake per ciascun canale può essere sinteticamente descritto come segue:

AXI defines a basic handshake mechanism, composed by an `xVALID` and `xREADY` signal. The `xVALID` signal is driven by the source to inform the destination entity that the payload on the channel is valid and can be read from that clock cycle onwards. Similarly, the `xREADY` signal is driven by the receiving entity to notify that it is prepared to receive data.

When both the `xVALID` and `xREADY` signals are high in the same clock cycle, the data payload is considered transferred and the source can either provide a new data payload, by keeping high `xVALID`, or terminate the transmission, by de-asserting `xVALID`. An individual data transfer, so a clock cycle when both `xVALID` and `xREADY` are high, is called "beat".



Two main rules are defined for the control of these signals:

- A source must not wait for a high `xREADY` to assert `xVALID`.
- Once `xVALID` is asserted, a source must maintain the assertion until a handshake occurs.

Thanks to this handshake mechanism, both the source and the destination can control the flow of data, throttling the speed if needed.

Facciamo riferimento al prototipo di AXI slave generato da Vivado e riportato in allegato (file AXISLAVE.pdf).

I process presenti, deputati alla gestione del protocollo, sono 9. Sono tutti sincronizzati con il clock, tranne il pocesso numero 08 (vedi file) che è asincrono.

E' il master a cominciare una operazione.

Processo di lettura

Il master mette su ARADDR l'indirizzo (porta S_AXI_ARADDR) e mette a 1 ARVALID (porta S_AXI_ARVALID). Il segnale ARPROT (porta S_AXI_ARPROT) è presente ma non utilizzato dallo slave, quindi lo ignoriamo.

Appena S_AXI_ARVALID diventa 1, entra in azione il processo 06 che mette a '1' il segnale axi_arready (collegato alla porta S_AXI_ARREADY) per un solo ciclo di clock, sfruttando il fatto che quando S_AXI_ARREADY diventa 1, il master mette a zero S_AXI_ARVALID al clock successivo. Contemporaneamente alla segnalazione di ready, viene memorizzato l'indirizzo di lettura nel segnale axi_araddr.

Sfruttando il segnale slv_reg_rden

slv_reg_rden <= axi_arready and S_AXI_ARVALID and (not axi_rvalid) ;

che è '1' solo nel ciclo di clock immediatamente successivo a quello in cui axi_arready va a 1 (axi_rvalid è normalmente '0' e diventerà '1' al circlo successivo) il process 8 (non sincronizzato) prepara il dato da mettere sul bus dati. Il process 9 utilizza slv_reg_rden per inviare questo dato su RDATA (vedi porta corrispondente). Parallelamente al process 9, il process 7 invia il segnale di dato valido e attende l'acknowledge da parte del master per ritornare nella condizione di riposo precedente all'inizio del processo di lettura.

Nota bene: il segnale slv_reg_rden è a 1 per un solo colpo di clock e il dato da leggere viene inviato sul bus contemporaneamente al ritorno a zero di questo segnale.

Si noti l'uso della variabile loc_addr nel process 8 che consente di avere l'immediato adeguamento di read_data in conseguenza del cambio di valore di axi_araddr.

Processo di scrittura.

Il processo di scrittura coinvolge i process da 01 a 05.

Il master mette su AWADDR (porta S_AXI_AWADDR) l'indirizzo di scrittura e segnala che è presente un indirizzo valido mettendo a '1' AWVALID (porta S_AXI_AWVALID). Allo stesso tempo (in realtà anche in tempi diversi: non cambia nulla) il master mette il dato da scrivere su WDATA (porta S_AXI_WDATA) e segnala che c'e' un dato valido mettendo a '1' WVALID (porta S_AXI_WVALID).

Nota che si usa WSTROBE per indicare quale dei byte della word è interessata dalla scrittura se non si sta scrivendo l'intero registro.

Non appena sia AWVALID sia WVALID sono ad '1', succedono le seguenti cose:

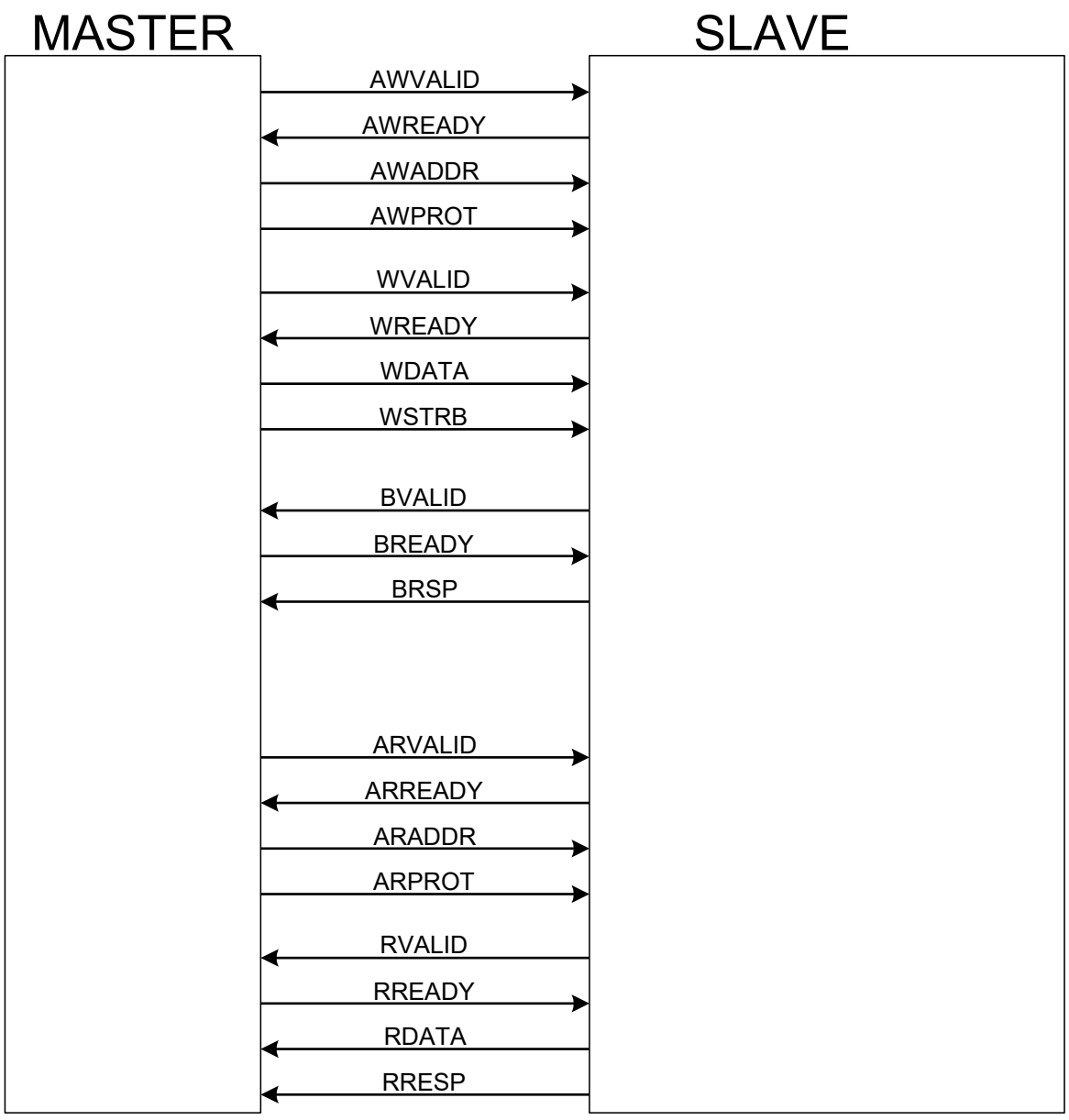
Il process 01 mette il segnale axi_awready (collegato a AWREADY, porta S_AXI_AWREADY) a '1' per un solo colpo di clock. Poi resta in attesa della conclusione dell'operazione di scrittura (S_AXI_BREADY='1') prima di tornare allo stato di partenza.

Nello stesso istante in cui il process 01 mette a 1 axi_awready, il process 02 memorizza in axi_awaddr l'indirizzo di scrittura.

Contemporaneamente, il process 03 segnala, mettendo a '1' la porta S_AXI_WREADY, che il dato è stato incamerato.

L'incameramento e la scrittura del dato nel registro slave (o parte di esso) avviene contemporaneamente all'azione del process 03 da parte del process 04.

Un colpo di clock dopo, il process 05 invia la "write response" ("00") e resta in attesa dell'accettazione da parte del master, così come fa il process 01, ovvero del ritorno a '1' del segnale S_AXI_BREADY, per ritornare in posizione di attesa di un nuovo ciclo di scrittura.



Write address channel	Write data channel	Write response channel	Read address channel	Read data channel
AWVALID	WVALID	BVALID	ARVALID	RVALID
AWREADY	WREADY	BREADY	ARREADY	RREADY
AWADDR	WDATA	BRESP	ARADDR	RDATA
AWPROT	WSTRB		ARPROT	RRESP