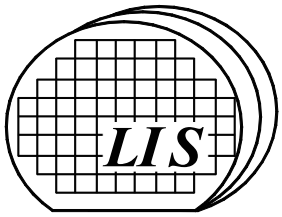
LIS-IPIF Specification

Johannes Zeppenfeld



Lehrstuhl für Integrierte Systeme
Arcisstraße 21
D-80290 München

TECHNISCHE
UNIVERSITÄT
MÜNCHEN

V1.3 – 21.7.2006

Table of Contents

1 Motivation

At the Institute for Integrated Systems (LIS) of the Munich University of Technology, we have several projects involving one or more user IPs connected to the Processor Local Bus (PLB) of a Xilinx FPGA, all of which currently use the Xilinx PLB IPIF (XP-IPIF) to initiate (master) and accept (slave) bus transfers. While this approach works fairly well for basic applications, the general consensus at LIS is that the XP-IPIF is a good deal slower than necessary, and that much of the performance available from the PLB is wasted due to the large latencies presented by the XP-IPIF. This study was therefore initiated to evaluate the XP-IPIF's performance, and to suggest the specification for a new PLB IPIF that resolves any deficiencies found.

The XP-IPIF was designed to be compatible with the Xilinx IP Interconnect (X-IPIC), which is less powerful than the interface provided by the PLB. This leads to an immediate decrease in the achievable performance of the XP-IPIF, the most obvious limitation being the shared data bus for both master and slave transactions. Given that the PLB is capable of simultaneous read and write transfers, it should be possible for the bus master unit of an IP to perform a write transfer while the slave of that same IP is performing a read. In fact, an efficient master or slave device should be capable of both a read and write transfer at the same time. Neither is supported by the XP-IPIF.

Since the X-IPIC is in effect its own bus, the XP-IPIF can be viewed as a bridge between the X-IPIC and PLB. A master request to the XP-IPIF is therefore translated into two unique transfers, one over the X-IPIC and one over the PLB. These transfers are performed sequentially, requiring that all data be buffered inside the XP-IPIF. Not only does this result in a large area overhead for storage, it also means that every transfer takes at least twice as long to complete. In addition, the entire XP-IPIF is blocked during master transactions, potentially resulting in long delays before its slave attachment can once again respond to requests made by other master devices on the PLB.

Maximum PLB utilization is achieved only when two dwords of data are transferred every single clock cycle, one in each direction (read and write). A few cycles are normally lost due to turnaround times when arbitration passes to a new master; however this can be eliminated by making use of the address pipelining mechanism provided by the PLB. The XP-IPIF does not support address pipelining, resulting in the additional loss of performance associated with arbitration turnaround.

Other limitations encountered in the XP-IPIF include an inability for the user IP to specify that further requests should be re-arbitrated. The additional transfer qualifiers available on the PLB are also missing from the X-IPIC interface (compress, guarded, ordered, etc.). Finally, the dual access transfer mechanism of the XP-IPIF does not allow for pure master IPs. Although any slave requests can be ignored in such cases, the logic required for the PLB slave attachment remains in the system, causing an unnecessary area overhead.

Taken together, the restrictions mentioned above provide a clear incentive for developing an optimized PLB to IP interface. Especially the dual access nature of the XP-IPIF should be avoided, allowing for simultaneous read and write transfers and preventing

slave access delays while the master is active. To avoid high latencies and large area requirements, minimal data buffering is another primary concern. In short, an IPIF implementation will be considered optimal only when it is able to correctly transfer one dword of data every clock cycle in both directions between a master and slave, and manages to do so with a negligible area overhead.

In Section 2, a complete performance analysis will be given to compare the latencies of XP-IPIF transfers with those projected for a new IPIF implementation. Design considerations for this new IPIF, henceforth referred to as the LIS-IPIF, will be presented in Section 3, resulting in a proposal for its specification, including a description of the signal interface provided to the IP, as detailed in Section 4. Finally, timing examples will be shown by means of sample waveforms in Section 5.

2 Performance Analysis

In order to evaluate the performance of the XP-IPIF, a rudimentary PLB test bench was constructed using the Xilinx PLB Arbiter. The resulting system was simulated with ModelSim to derive the latencies of both read and write bus transfers using various combinations of master and slave attachments. A behavioral model of the LIS-IPIF was used to verify the feasibility of the timings presented in Section 5, and to provide a context for the results obtained with the XP-IPIF.

Since the Xilinx Arbiter uses a three-cycle arbitration scheme instead of the single cycle arbitration theoretically possible over the PLB, all latency measurements were taken in such a way as to avoid the additional two cycle arbitration overhead. In addition, latency measurements for slave and master devices were taken independently. Slave latency was therefore measured from the clock cycle in which PLB_PAValid is asserted through the last cycle of the data transfer. Master transfers are split into two parts, the first latency being the number of cycles needed for a request to propagate from the IP to the PLB. The second latency indicates the number of cycles required before the last data acknowledge becomes visible to the IP.

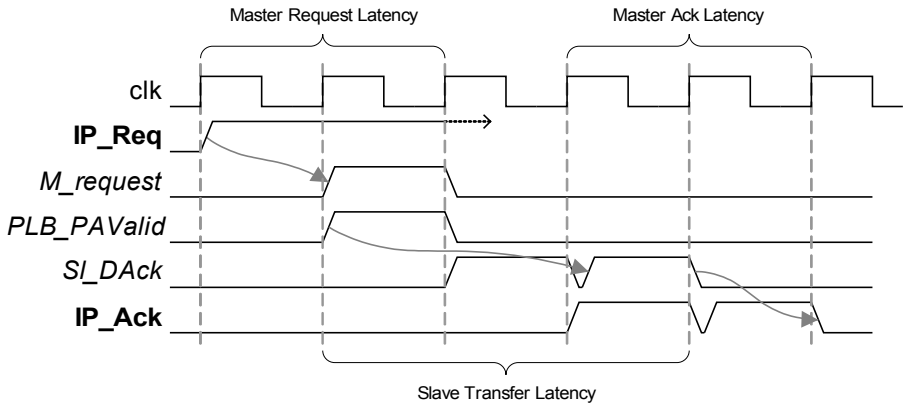


Figure 1: Latency Measurements

	Xilinx IPIF		*LIS IPIF*		*Optimal*	
Burst Size	***Rd***	***Wr***	***Rd***	***Wr***	***Rd***	***Wr***
1	5	5	3	1(+1)	3	1
2	7	8	4	2(+1)	4	2
4	11	14	6	4(+1)	6	4
8	19	26	10	8(+1)	10	8
16	35	50	18	16(+1)	18	16
32	67	98	34	32(+1)	34	32
x	2·x+3	3·x+2	x+2	x(+1)	x+2	x

Table 1: Slave Latency Comparison

Burst Size	***Xilinx IPIF***				***LIS IPIF***				***Optimal***	
	Rd-Req	***Rd-Ack***	***Wr-Req***	***Wr-Ack***	***Rd-Req***	***Rd-Ack***	***Wr-Req***	***Wr-Ack***	***Rd***	***Wr***
1	8	5(+1)	10	2	2	2	2	2	1,1	1,1
2	8	-*	11	-*	2	2	2	2	1,1	1,1
4	8	8(+1)	13	2	2	2	2	2	1,1	1,1
8	8	12(+1)	17	2	2	2	2	2	1,1	1,1
16	8	20(+1)	25	2	2	2	2	2	1,1	1,1
32	-	-	-	-	2	2	2	2	1,1	1,1
x	8	4+x(+1)	5+x+4	2	2	2	2	2	1,1	1,1

Table 2: Master Latency Comparison

The two tables above show the read and write cycle latencies for both XP-IPIF and LIS-IPIF. The minimum latency required according to the PLB specification is also provided for comparison. Whereas the proposed latencies for the LIS-IPIF nearly match the optimal values, the latencies of the XP-IPIF are much larger.

Looking first at the results obtained for the slave attachment, it can be seen that the XP-IPIF requires nearly twice as long for read and three times as long for write transfers as necessary according to the optimal values. This can be attributed to the fact that acknowledge-signals maintain a strict causality in addition to being passed through registers designed to isolate the IP from the PLB, as shown in Figure 2. During read transfers, the falling edge of the burst signal indicates the burst's completion. Assuming the slave were capable of returning data every clock cycle (shown by the dashed Ack pulse), the burst signal wouldn't go low until after the last data dword was requested from the IP. The slave request signal would then have to be negated in the same cycle that the burst signal goes low, resulting in a purely combinational path between PLB and IP. During write transfers the latency can be attributed solely to the flow of request and acknowledge signals. An additional cycle is needed in this case for the arriving data to be buffered before being presented to the IP.

* A bug in the XP-IPIF causes a transfer of the wrong size. (Read 34, Write ∞?)

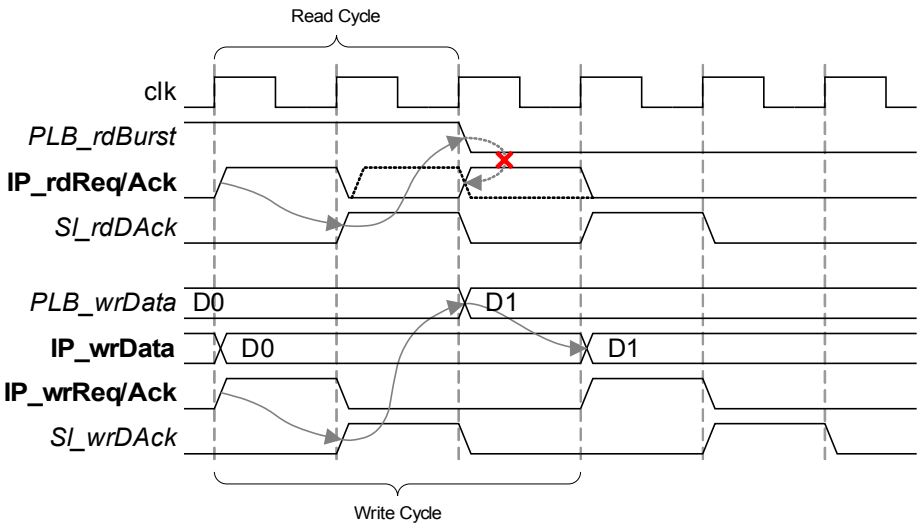


Figure 2: Read/Write Slave Causality

Moving on to the master attachment, the dual access nature of the XP-IPIF is clearly demonstrated by the dependence on burst size exhibited by read-acknowledges and write-requests. During a read transfer, all acknowledged data is stored inside the IPIF before being passed to the IP. Similarly during write transfers, a PLB request does not occur until the complete data packet has been buffered within the IPIF. This makes some sense during write transfers to ensure that the full data packet is available for transmission across the PLB (once a PLB-transfer has been started, there is no method for the master to throttle the burst until completion), but there is no reason not to provide incoming data directly to the IP during read cycles. The fixed cycle costs are also much larger than necessary, probably due to the arbitration of the X-IPIC between master and slave attachments within the XP-IPIF.

3 Design Considerations

In order to assure that the LIS-IPIF is not simply a clone of the XP-IPIF, a priority of objectives must be chosen to distinguish the two implementations from each other. Judging by the results obtained in the previous section, the primary goals of the XP-IPIF were rigorous fault tolerance (as exhibited by the strict req/ack flow), full separation of PLB and IP (complete data buffering within the IPIF) and maximum compatibility with existing IPs (use of the X-IPIC), all at the cost of performance. The focus of the LIS-IPIF shall therefore be to maximize performance instead, making slight concessions in other areas as necessary to allow for the lowest possible latency.

Although increased performance is the primary goal of the LIS-IPIF, other design criteria presented by the XP-IPIF should not be disregarded. Data buffering within the IPIF is therefore maintained in order to ensure a separation of logic between PLB and IP. Rather than storing the complete data transfer within the IPIF however, only a small part of the transfer shall be buffered at a time, greatly reducing the required area overhead. By overlapping the PLB and IP data transfers, the large latencies resulting from two sequential transactions are also avoided.

Compatibility with existing IPs through support of the X-IPIC or other standard bus interface would be a nice feature, but the resulting overheads are simply not acceptable for a performance optimized PLB IPIF. A new bus interface is therefore proposed that completely separates the master and slave devices, and additionally allows for simultaneous read and write transfers. The complete specification for this bus is provided in Section 4.2.

In order to minimize the latency of a PLB transfer, it must be possible to transfer one dword of data each and every clock cycle while a burst is in progress. This is used as the basic requirement that must be fulfilled by the LIS-IPIF. Revising the waveforms of Figure 2 as shown in Figure 3, it becomes apparent that a few limitations must be accepted within the slave attachment for this to be possible. During read transfers, data must be requested from the IP one cycle in advance of being presented to the PLB. This means that the IP read request signal must be negated within the same clock cycle that the read burst signal from the PLB goes low, resulting in a combinatorial dependence between PLB and IP. During write transfers, incoming data must be acknowledged in the same clock cycle it arrives from the PLB. Since the data cannot be presented to the IP until the following clock cycle due to data buffering, the IPIF must acknowledge the data before having made sure that the IP will accept it.

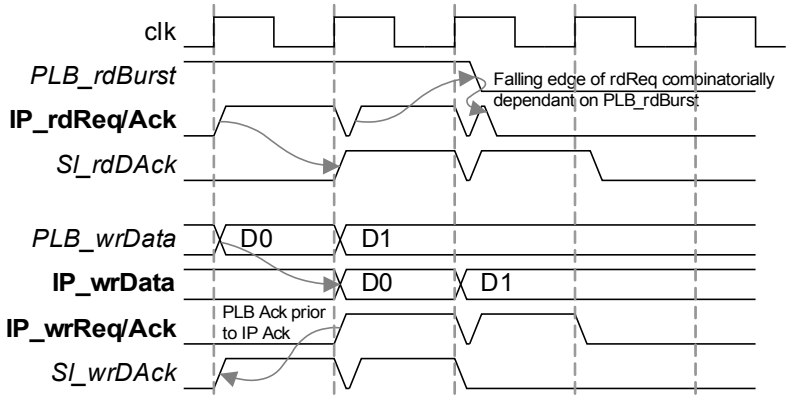


Figure 3: Necessary Limitations of the LIS-IPIF Slave

There are several possibilities for resolving the acknowledge causality issue during write transfers. The first is to revert to the request/acknowledge flow used by the XP-IPIF, which is not admissible due to its insertion of two additional clock cycles. Instead, it is possible to have the user IP provide an early acknowledge signal, indicating that a request in the next clock cycle is guaranteed to be acknowledged. The PLB write data acknowledge would have to be combinatorially dependant on this early acknowledge however, nullifying any merits of the solution. An even earlier acknowledge two cycles in advance seems unreasonable despite potentially resolving the dependency issue.

No solution has been found thus far to resolve the combinatorial dependence exhibited during read transfers, other than inserting an additional clock cycle as in the XP-IPIF. Making the user aware that the read request signal may be negated fairly late in the clock cycle at the end of a burst transfer must therefore suffice. Note that this is only an issue during variable length bursts, where the burst size is not known from the start!

In order to increase performance and reduce the area overhead, buffering of burst data within the master attachment of the LIS-IPIF is minimized. Since a PLB master attachment is incapable of throttling burst transfers once they have begun, this requires that the user IP be able to accept (read) and deliver (write) a complete data dword every clock cycle while a burst is in progress. This is the case by default in many IP designs anyways, so adding additional buffers within the IPIF is often redundant. For those IPs not capable of a one data per clock throughput, the user can always insert a FIFO or similar buffer design between IP and LIS-IPIF to provide the necessary data buffering externally.

Arbitration overheads on the PLB can be completely eliminated through the use of address pipelining within the slave attachment. This potentially allows both read and write PLB data buses to constantly transfer data. Although the latency reduction made possible by address pipelining involves only very few clock cycles, for single dword or very short burst transfers this overhead can make a significant difference in overall performance. Pipelining should therefore be supported by the LIS-IPIF, possibly as an option that allows the user to choose between reduced performance and the area overhead required for pipelining.

Further PLB functionality to be supported by the LIS-IPIF includes bus re-arbitration when the slave is busy. This allows a busy user IP to prevent the bus from being blocked for long periods of time. The ability for an IP master to lock the PLB should also be provided by the LIS-IPIF master attachment. Finally, any transfer qualifiers not supported by the IPIF must be made available for processing in the IP itself, making it possible for the user IP to extend the functionality of the IPIF without making modifications to the IPIF itself.

4 Specification

Based on the design considerations given in the previous section, a specification for the proposed LIS-IPIF is presented in the following. As already mentioned, the key goal to be achieved is maximized performance with a small area overhead. Other criteria should be optimized where possible, so long as they don't conflict with the primary objectives.

4.1 LIS-IPIF Functionality

4.1.1 Separation of Master and Slave

The PLB treats master and slave devices as completely separate entities, and the LIS-IPIF will be implemented accordingly. This allows for IPs with either master or slave attachments, and also for IPs with both. In addition, both the master and slave attachment may be active at the same time.

Since the PLB allows for simultaneous read and write transfers, the master and slave units of the LIS-IPIF provide the user IP with two similar but separate interfaces (sub-buses), one for read and the other for write transactions. The slave automatically routes all incoming requests to the appropriate sub-bus based on the transfer direction. This cannot be achieved so easily within the master, since the PLB provides only a single arbitration bus that must be shared between read and write transfers. The LIS-IPIF master

is therefore responsible for arbitrating between read and write requests made by the user IP, using a configurable read-priority, write-priority or round-robin arbitration scheme. As this arbitration must take place between only two entities (read and write), none of the presented schemes are overly complex.

With a separate bus being provided for read and write transfers on both the master and slave attachment, a user IP with master and slave capabilities has up to four buses that may require simultaneous access to a single resource. In order to resolve the resulting conflict, the user IP may assert the busy signal of any of its LIS-IPIC interfaces to ensure that requests to the same resource don't overlap. See Section 4.1.8 for more information.

4.1.2 Transfer Types

The LIS-IPIF master attachment equally supports both read and write transfers, each of which can be for either an individual dword using byte enables, or for a burst of several dwords. The length of the burst must be specified along with the request, and may be of any size. Byte enables are ignored whenever the transfer length is greater than one.

In the case of a single dword transfer, the PLB specification requires that the address passed onto the bus by the master must address the leftmost byte as indicated by the byte enable. If the user does not wish to implement this functionality, recalculation of the least significant address bits can be optioned into the LIS-IPIF. Removing this option for compliant IPs will result in a reduction of IPIF area.

Whereas unsupported transfer types within the LIS-IPIF master are simply never initiated, the slave attachment should be able to accept all types of incoming requests, including single, line and burst transfers of varying data widths. Until byte steering is implemented in the LIS-IPIF (see Section 4.1.4 below), and to promote maximum bus utilization, burst transfers with a data width of less than 64 bits will not be supported, and any such requests made to the slave will not be acknowledged.

Although several different types of transfer are supported by the LIS-IPIF slave, they all appear nearly identical to the user IP. Burst transfers are classified into known- and variable length, depending on whether the number of dwords to be transferred is available from the start. All line transfers are presented to the IP as known-length bursts. Single dword transfers are different only in that the byte enable signal may potentially contain zeroes. Otherwise they appear as bursts with a known length of one.

4.1.3 Address Decoding

Every slave is responsible for monitoring the PLB's address bus and responding to any requests that target the slave's address range. For proper integration into the Xilinx EDK design flow, this range is provided to the slave as two generics, indicating a base and high address. The LIS-IPIF slave will acknowledge any requests in which the most significant bits of the address match those supplied by the generics.

During large burst transfers near the end of the address range, an overflow into addresses outside of the slave's range may occur. Since this is in effect a transfer error originating in the master, adding logic to detect this overflow should not be a burden placed on the

slave. If the user wishes to provide overflow detection anyways, they can do so within the IP itself, terminating the transfer should it prove necessary (see Section 4.1.8).

Due to the large number of possible addressing requirements within various slave user IPs, automated address incrementing within the IP is provided as an option. This allows any slave with its own addressing logic to avoid incrementor redundancy and the associated area overhead. If address incrementing is optioned out, only the starting address of a burst will be provided to the IP slave, after which the IP must provide data sequentially until the complete signal is asserted by the IPIF. The next request will again be made with a valid address.

4.1.4 Data/Byte Steering

In order to reduce the complexity of the initial LIS-IPIF release, all byte-steering logic theoretically required for 64-bit PLB devices will be omitted, meaning that the LIS-IPIF may only access or be accessed by other 64-bit devices. Furthermore, the width of the data buses leading to/from the user IP must match the data width of the PLB. This eliminates the necessity for data mirroring in the master and data steering within the slave, greatly reducing the complexity of the IPIF. Conversion cycles required when a 64-bit master accesses a 32-bit slave can also be avoided.

Since future applications involving the LIS-IPIF may require both 32- and 64-bit devices on the same bus, it should remain possible to add all functionality required for mixed width environments with only minor modifications to the rest of the LIS-IPIF. This potential extension must therefore be kept in mind during the initial design in order to facilitate an upgrade later on. User logic with a data width less than that of the PLB will not be considered, as the required byte steering can easily be performed by the IP in such cases.

Whereas all bus signals on the PLB are indexed using “low to high” notation, the LIS-IPIC conforms to the LIS standard of using “high downto low” for all bit vectors, as shown in Figure 4 by way of a data bus. Due to the big-endian nature of the PLB, this means that a byte address of zero corresponds to the bits numbered 63 downto 56, address one corresponds to the bits 55 downto 48 and so on. This is especially important to consider when accessing packed registers in an IP, where multiple registers may be located within the same 64-bit dword. In such cases the register with the lowest address will be situated at the highest bit-index of the dword.

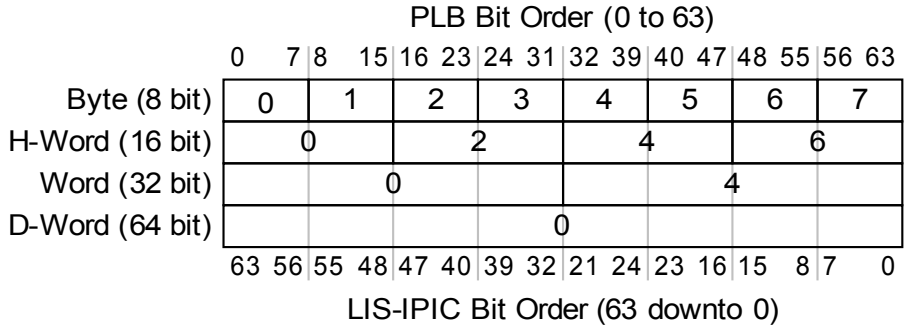


Figure 4: Bit Reordering and Addressing within the LIS-IPIF

4.1.5 Data Buffering

Providing a complete stage of signal buffering between the PLB and IP frees the user from having to consider the timing requirements of the PLB specification. In addition, the user may rely on the accuracy of the timing reports provided by the synthesis tools, without worrying about the timing of signals beyond the IPs I/O ports.

To reduce the area overhead required for buffering of large burst transactions, only a few dwords of the transfer will be stored within the IPIF at any given time. Since the master has no way of throttling a burst once it has begun, the user IP is responsible for accepting or providing data at the same rate it is transferred across the PLB, up to the maximum of one dword per clock. IPs not capable of this data rate must implement an additional FIFO or similar storage device at their outputs, where data can be buffered before being passed to the IPIF.

As discussed in Section 3, the LIS-IPIF slave's read request signal must be negated in the same clock cycle that the PLB's read burst signal goes low. This is the only case of a combinatorial dependence between PLB and IP. According to the PLB specification, the read burst signal is valid within 43% of the clock cycle. Adding in a small delay for routing and simple logic within the IPIF, the read request signal may not go low until 50% of the clock cycle after the last transfer. This is the case only at the end of variable length burst transfers, in which the transfer size is not known from the start.

4.1.6 Address Pipelining

Maximum data throughput on the PLB is made possible by allowing for arbitration of a new transfer while the previous transfer is still in progress. This is accomplished through address pipelining, which is fully supported by the LIS-IPIF and allows for constant activity on both the PLB and LIS-IPIC.

Although any slave pipelining is performed transparently to the user IP, the back to back nature of incoming requests requires an additional signal that indicates when one request completes and the next request begins. This is accomplished with a "complete" signal passed to the IP, which indicates that the currently active transfer has finished. Any request made in the next cycle will belong to a new transfer, and may potentially involve an address discontinuity. It should be noted that the complete signal is not necessarily asserted along with the last item of data, but may be asserted one or more clock cycles thereafter.

The PLB's separation of arbitration and data buses can be made available to the user IP master by enabling pipelining within the LIS-IPIF master attachment. This permits the IP to request additional transfers before the previous transfer(s) have been completed, and allows full bus utilization by a single master. When master pipelining is enabled, additional requests can be made immediately after the previous request has been accepted. Without pipelining, further requests won't be accepted until after the data transfer has completed. User IPs supporting master pipelining must ensure that their request signal is negated in the clock cycle after it has been accepted by the IPIF, unless an additional request is being made. In that case, the qualifiers for the new transfer must be made available along with the request.

4.1.7 Bus Locking

The master IP may lock the PLB by asserting the lock signal along with a request. When a request is accepted by the LIS-IPIF with the lock signal asserted, no transfers from other PLB masters in either direction will be interspersed until the master IP has negated the lock signal. Multiple transfers may be requested while the lock signal is held high, all of which will be passed to the PLB atomically. Note that all other masters on the PLB will be starved during this time, so bus locking should be used sparingly.

4.1.8 Transfer Rearbitration / Termination

If an IP slave is busy performing an operation and temporarily cannot respond to additional incoming transfers, it should assert the busy signal associated with one or both transfer directions (read or write). All further PLB requests in that direction will then be re-arbitrated until the busy signal is negated. Additionally, any incoming pipelined (secondary) requests will be ignored during this time, or be re-arbitrated should they become primary requests before the busy signal goes low.

When a slave requests for a LIS-IPIF master to re-arbitrate the current transfer, the user IP is informed by the assertion of the rearbitrate signal. If no further actions are taken by the IP, the LIS-IPIF will hold the request indefinitely until accepted by the slave. To prevent this, the IP master may assert its abort signal to withdraw the request, in which case the IP is responsible for repeating the request at a later time (should it be necessary).

In addition to requesting re-arbitration of a transfer, both the master and slave may assert the abort signal throughout a transfer to terminate that transfer prematurely. In the case of a pipelined transfer, the abort signal will always apply to the data transfer currently in progress. Since any data already on the bus while an abort request is made must reach its destination, additional transfer cycles may be necessary as shown in Table 3. This only affects aborts requested by the unit which is receiving data. An abort requested along with the last data item in a transfer is ignored.

	Read	***Write***
Master	2	0
Slave	0	2

Table 3: Maximum additional Transfers after Abort

4.1.9 Error Handling

The PLB provides very few error handling capabilities beyond those necessary to re-arbitrate or terminate a transfer. Instead, any error and status information is intended to be exchanged over the device control register (DCR) bus. A connection to this bus is not provided by the LIS-IPIF, and must be implemented separately within the user IP (if desired).

The LIS-IPIF supports both re-arbitration and transfer termination as discussed in Section 4.1.8 above. When a transfer is terminated by a slave, the PLB error signal to the associated master will be asserted. Each slave IP may additionally trigger this error signal by asserting its own error signal, without aborting the transfer. Please note that due to the

error handling limitations of the PLB, it may not be possible for the master to determine which slave is asserting the error signal without DCR-bus support. Any errors arriving at the LIS-IPIF master attachment will be passed through to the user IP, and no additional error processing is performed by the IPIF.

4.2 LIS-IPIC Bus Interface

	Master				Pg.	***Slave***				Pg.
Read	rdAddr	I	32	Source address	14	rdAddr	O	32	Request address	15
	rdBE	I	8	Data byte enables	15	rdBE	O	8	Data byte enables	15
	rdData	O	64	Retrieved data	16	rdData	I	64	Outgoing data	16
	rdNum	I	≥5	Transfer Length	17	rdNum	O	5	Transfer length	17
	rdComp	O	-	Transfer Complete	16	rdComp	O	-	Transfer complete	16
	rdReq	I	-	Read Request	18	rdReq	O*	-	Read request	18
	rdAck	O	-	Data available	14	rdAck	I	-	Data ready at output	14
	rdAccept	O	-	Request accepted	14	rdErlyAck	I†	-	Early acknowledge	17
	rdPriority	I†	2	Master priority	20	rdMaster	O	≤4	Requesting master	19
	rdType	I†	3	Request type	20	rdType	O	3	Request type	20
	rdCompress	I†	-	Transfer qualifier	19	rdCompress	O	-	Transfer qualifier	19
	rdGuarded	I†	-	Transfer qualifier	19	rdGuarded	O	-	Transfer qualifier	19
	rdLockErr	I†	-	Transfer qualifier	19	rdLockErr	O	-	Transfer qualifier	19
	rdRearbitrate	O	-	Rearbitrate request	22	rdBusy	I†	-	Rearbitrate requests	21
	rdAbort	I†	-	Abort transfer	20	rdAbort	I†	-	Abort transfer	21
	rdError	O	-	Read error occurred	21	rdError	I†	-	Indicate error	
Write	wrAddr	I	32	Target address	14	wrAddr	O	32	Request address	15
	wrBE	I	8	Data byte enables	15	wrBE	O	8	Data byte enables	15
	wrData	I	64	Output data	16	wrData	O	64	Incoming data	16
	wrNum	I	≥5	Transfer length	17	wrNum	O	5	Transfer length	17
	wrComp	O	-	Transfer complete	16	wrComp	O	-	Transfer complete	16
	wrReq	I	-	Write request	18	wrReq	O	-	Write request	18
	wrAck	O	-	Data acknowledged	14	wrAck	I	-	Data acknowledge	14
	wrAccept	O	-	Request accepted	14	wrErlyAck	I†	-	Early acknowledge	17
	wrRdy	O	-	Data accepted	17					
	wrPriority	I†	2	Master priority	20	wrMaster	O	≤4	Requesting master	19
	wrType	I†	3	Request type	20	wrType	O	3	Request type	20
	wrCompress	I†	-	Transfer qualifier	19	wrCompress	O	-	Transfer qualifier	19
	wrGuarded	I†	-	Transfer qualifier	19	wrGuarded	O	-	Transfer qualifier	19
	wrOrdered	I†	-	Transfer qualifier	20	wrOrdered	O	-	Transfer qualifier	20
	wrLockErr	I†	-	Transfer qualifier	19	wrLockErr	O	-	Transfer qualifier	19
	wrRearbitrate	O	-	Rearbitrate request	22	wrBusy	I†	-	Rearbitrate requests	21
	wrAbort	I†	-	Abort transfer	20	wrAbort	I†	-	Abort transfer	21
	wrError	O	-	Write error occurred	21	wrError	I†	-	Indicate error	22
	Error	O	-	Error occurred	21					
	Lock	I†	-	Lock bus	22	Locked	O	-	Bus is locked	22

Table 4: Master (left) and Slave (right) Interface Signals

* Combinatorially dependent on PLB_rdBurst.
† Optional, drive low if not used.

4.2.1 Request Arbitration

4.2.1.1 M_rdAccept, M_wrAccept (Transfer Accepted)

This signal is asserted by the master attachment of the LIS-IPIF to indicate that the associated request (read or write) being made by the IP will be latched at the end of the current clock cycle, including all transfer qualifiers. If pipelining is enabled within the LIS-IPIF master, additional requests may be accepted before the transfers associated with previous requests have completed. To prevent the same transfer from being repeated multiple times, the user IP must therefore negate its request signal in the clock cycle following its acceptance. Non-pipelined masters may keep their request signal asserted until the clock cycle in which the complete signal goes high.

- Asserted by the LIS-IPIF in response to M_xxReq.

4.2.1.2 M_rdAck, M_wrAck (Master Data Acknowledge)

This signal is provided to the IP master to indicate that the associated dword of data has been acknowledged by the slave. During a read transfer, the data on the M_rdData bus is valid while this signal is asserted. Any data transferred during a write request is provided to the LIS-IPIF before being acknowledged (see page 17 on the M_wrRdy signal), and the M_wrAck signal only indicates that that data has actually been accepted by the slave.

After the IP master has requested a transfer, the associated acknowledge will be asserted once for each data item according to the size specified by M_xxNum, unless the transfer is aborted prematurely by either the master or slave. In this case the M_xxComp signal will be asserted along with the M_xxError signal prematurely, and any additional acknowledges will be for a subsequent transfer.

- Asserted by the LIS-IPIF when data is available from (read) or has been acknowledged by (write) the slave.

4.2.1.3 S_rdAck, S_wrAck (Slave Data Acknowledge)

This signal is asserted by the slave IP to indicate that the data of the current transfer is available (reads) or has been accepted (writes). It may be asserted within the same clock cycle that the corresponding request signal becomes active, or any clock cycle thereafter.

- Asserted by IP in response to S_xxReq. Must be negated when S_xxReq is low!

4.2.1.4 M_rdAddr, M_wrAddr (Master Address)

The address bus specifies the target address of the current request, and must fall within the address range of one of the slaves located on the PLB. If no slave responds to the specified address, the transfer will timeout with the LIS-IPIF asserting both the M_xxComp and M_xxError signals, without any data being acknowledged.

During single dword transfers using partial byte enables, the address presented to the PLB must always refer to the most significant byte of enabled data, as shown in Table 5. If this is not supported by the user IP, recalculation of the least significant address bits can be enabled within the LIS-IPIF using the C_EN_RECALC_ADDR generic.

M_xxBE (7:0)	*M_xxAddr (2:0)*
1xxxxxxx	000
01xxxxxx	001
001xxxxx	010
0001xxxx	011
00001xxx	100
000001xx	101
0000001x	110
00000001	111

Table 5: Dependence of Address on Byte Enables

- Must be valid whenever M_xxReq is asserted.

4.2.1.5 S_rdAddr, S_wrAddr (Slave Address)

This bus indicates the starting address of each transfer to the slave, or the address of every individual request if address incrementation is enabled within the IPIF by use of the C_EN_INC_ADDR generic. During single item data transfers, the three low order bits of the address will indicate the most significant enabled byte, as shown in Table 5 above. These bits will always be zero during burst transfers.

Although the initial address provided to the user IP during a burst transfer is guaranteed to fall within the slave's address range, no overflow detection is performed by the IPIF. The slave may prevent this by monitoring the address and aborting the transfer prior to an overflow. If address incrementation is enabled, this range checking can also be performed within the IPIF by setting the C_ENFORCE_RANGE generic. In this case the transfer will be aborted in time to prevent an overflow.

- C_EN_INC_ADDR enabled: Valid any time S_xxReq is asserted.
- C_EN_INC_ADDR disabled: Valid along with the first S_xxReq of each transfer.

4.2.1.6 M/S_rdBE, M/S_wrBE (Byte Enable)

These signals identify which bytes of the dword currently being presented on the corresponding data bus contain valid data. Byte enables are used only during single dword transfers, and will be driven high or ignored while a burst is in progress. Although not specifically required by either the master or slave attachment of the LIS-IPIF, the PLB specification requires that there be no gaps between the enabled bytes of data. The behavior when a master asserts the byte enables of only the most and least significant bytes of data (for example) is therefore undefined.

- Master: Must be valid whenever M_xxReq is asserted.
- Slave: Valid any time S_xxReq is asserted.

4.2.1.7 M_rdComp, M_wrComp (Master Transfer Complete)

This signal is asserted by the master attachment of the LIS-IPIF to indicate that the current data transfer has completed or will complete by the end of the current clock cycle. It will be asserted along with or after the last data acknowledge of the transfer, but at least one clock cycle prior to any acknowledge from a following transfer.

When a burst transfer has been aborted by the master or slave, the complete signal may be asserted before the requested amount of data has been transferred. In this case the associated error signal will be asserted along with the complete signal, and the master is expected to terminate the transfer by the end of the clock cycle.

During pipelined write operations, data for the following transfer may be passed to the LIS-IPIF in response to the M_wrRdy signal before the complete signal is asserted. This is not an issue during pipelined reads, since data arrives synchronously with the acknowledge signal. See page 17 on the M_wrRdy signal for details.

- Asserted to indicate completion of a transfer, negated at all other times.

4.2.1.8 S_rdComp, S_wrComp (Slave Transfer Complete)

This signal is asserted by the LIS-IPIF to indicate that the current data transfer has completed, or will complete with the next acknowledged request. When asserted along with the last data request of a transfer, it will remain asserted until the IP slave has acknowledged that data. If asserted while S_xxReq is negated, it will remain high for only a single clock cycle. Any request made in a following cycle will belong to a subsequent transfer.

- Asserted by the LIS-IPIF at the end of each transfer.

4.2.1.9 M_rdData, S_wrData (Data In)

These 64-bit data buses provide data from the LIS-IPIF to the user IP. All data provided on these buses is aligned so that the most significant byte of data corresponds to the address in which the three low order bits are negated (i.e. big endian). Any bytes on the data bus not validated by the associated byte enable signal should be ignored, and not be committed to the underlying memory structure of the IP.

- M_rdData is valid while M_rdAck is asserted.
- S_wrData is valid while S_rdReq is asserted.

4.2.1.10 M_wrData, S_rdData (Data Out)

These 64-bit data buses are used to transfer data from the user IP to the LIS-IPIF. Any data presented on these buses must be properly aligned, in that the most significant byte of data corresponds to the address in which the three low order bits are negated (i.e. big endian). Byte enables may be disregarded when driving data onto these buses, so long as those portions of data with enabled bytes are valid.

- M_wrData must be valid for the entire transfer duration.
- S_rdData must be valid whenever S_rdAck is asserted.

4.2.1.11 S_rdErlyAck, S_wrErlyAck (Early Data Acknowledge)

When asserted by the user IP slave, this signal indicates that any request made in the next clock cycle will be acknowledged by the slave within that cycle. This is used by the LIS-IPIF to generate the complete signal one cycle prior to the last data transfer, which allows for a faster turnaround between successive transfers. A slave that can always return data within the same clock cycle it is requested may tie this signal high. Slaves that cannot make this guarantee may tie the signal low, causing a one-cycle performance penalty between transfers.

- Must be valid at all times.

4.2.1.12 M_rdNum, M_wrNum (Master Transfer Length)

This bus indicates the number of dwords that should be transferred to complete the associated request. Any transfer length greater than one will initiate a burst transfer to the addressed slave, whereas a single dword transfer will be requested otherwise. A request made for bursts of no more than 16 dwords will be translated by the LIS-IPIF into a fixed length burst of the appropriate size*. Longer transfers are made using variable length bursts, in which case the slave cannot be informed of the transfer length in advance. It is therefore recommended to use a maximum burst length of 16 dwords (128 bytes), concatenating multiple bursts as necessary to complete the transfer.

- Must be valid any time M_xxReq is asserted.

4.2.1.13 S_rdNum, S_wrNum (Slave Transfer Length)

During known length bursts, this bus informs the user IP slave of the number of data items to be expected for the current transfer, and will be decremented by one with each item of data that is acknowledged by the slave. During single dword transfers this value will be set to one, whereas it will be set to zero during variable length transfers in which the transfer length is not known from the start.

If a known-length transfer is aborted prematurely, the complete signal may be asserted before the number of remaining transfers reaches zero. In this case the remaining data count as indicated by S_xxNum should be ignored, and the transfer aborted by the end of the cycle in which S_xxComp was asserted. S_xxNum will become valid again along with the next S_xxReq.

- Valid any time S_xxReq is asserted, as noted above.

4.2.1.14 M_wrRdy (Ready for Data)

During write transfers, this signal is asserted by the LIS-IPIF to indicate that the data on the M_wrData bus has been accepted, and that the user IP must provide the subsequent dword of data in the following clock cycle. The actual slave's acknowledgement of the data will occur in a future clock cycle by means of the M_wrAck signal.

It is important to note that the ready signal does not guarantee that the associated dword of data will be acknowledged by the slave, but rather that the LIS-IPIF has accepted the

* Impl. Possibility: Only allow predetermined width bursts (length <= 16).

data and will attempt to transfer it to the slave as quickly as possible. In fact, the data ready signal may be asserted as early as the clock cycle in which a request made by the IP is accepted, before that request has even been presented to the PLB arbiter. The data will not have been passed to the slave until the corresponding acknowledge has been asserted.

When pipelining is enabled within the LIS-IPIF master attachment, the first data ready of a write transfer may be asserted before the previous transfer has completed. In this case, the user IP is responsible for ensuring that the proper data of the new transfer is being presented on the data bus, while also waiting for the previous transfer to complete.

If a transfer is aborted prematurely, any remaining data for that transfer which has been accepted by the LIS-IPIF will be flushed. Data that has already been accepted for the following transfer will be preserved, allowing for a continuous data flow in response to the data ready signal. If a transfer is aborted before all of its data has been accepted, the user IP must begin to provide data for the following transfer in the clock cycle after the assertion of M_wrComp.

- Valid throughout the entire transfer.

4.2.1.15 M_rdReq, M_wrReq (Master Request)

This signal should be asserted by the user IP master to request a new transfer with the addressed slave. If the master is pipelined, this signal must be negated in the clock cycle after M_xxAccept is asserted, unless an additional transfer is to be requested. When master pipelining is disabled, the IP may hold the request until the end of the clock cycle in which M_xxComp is asserted.

- May be asserted any time all transfer qualifiers are valid.

4.2.1.16 S_rdReq, S_wrReq (Slave Request)

This signal is asserted by the slave attachment of the LIS-IPIF to indicate that a master is requesting a read or write from the slave. This signal will remain asserted until the slave has acknowledged the requisite number of data dwords, unless the transfer is aborted prematurely. A transfer is considered complete after the clock cycle in which S_xxComp is asserted.

As discussed in Section 4.1.5, the S_rdReq signal exhibits a combinatorial dependence on PLB_rdBurst. In order to minimize the impact this has on LIS-IPIF performance, only the falling edge of the S_rdReq signal will be influenced, and only at the end of variable length bursts. During single dword transfers, or when the burst length is known from the start, S_rdReq will be properly negated close to the beginning of the clock cycle. In addition, the slave will remind the master that the transfer should be ended by asserting the PLB terminate signal.

- Asserted for the entire transfer, negated when idle.

4.2.2 Transfer Qualifiers

4.2.2.1 M/S_rdCompress, M/S_wrCompress (Compressed Transfer)

This signal is asserted by the master to indicate that the target memory location contains compressed data*. During a write transfer, the slave is responsible for compressing any data provided by the master before storing it to memory. For a read transfer, the slave must decompress the data in memory prior to returning it to the master. See the PLB specification for details on the associated PLB_compress signal.

- Valid any time M/S_xxReq is asserted.

4.2.2.2 M/S_rdGuarded, M/S_wrGuarded (Guarded Transfer)

This signal is asserted by the master to indicate that the current transfer may be for a non-well behaved memory, and that the slave should limit its memory accesses to those requested by the master. Consult the PLB specification for details and the intended use of guarded transfers and the associated PLB_guarded signal.

- Valid any time M/S_xxReq is asserted.

4.2.2.3 M/S_rdLockErr, M/S_wrLockErr (Lock Errors)

This signal is driven by the master to indicate that any errors occurring during the current transfer must be locked by the slave's error status register (SESR) and error address register (SEAR). See the PLB specification for details on the associated PLB_lockErr signal.

- Valid any time M/S_xxReq is asserted.

4.2.2.4 S_rdMaster, S_wrMaster (Master ID)

The master ID bus indicates to the slave which master requested the current transfer. The width of this bus is dependant on the number of masters present on the PLB, as shown in Table 6, and must be provided to the LIS-IPIF by means of the C_PLB_MID_WIDTH generic.

Number of Masters	*S_xxMaster Width*
2	1
3 to 4	2
5 to 8	3
9 to 16	4

Table 6: S_xxMaster Bus Width

- Valid any time S_xxReq is asserted.

* When used in conjunction with the embedded PowerPC of a Virtex II FPGA, this signal instead represents the user-defined storage attribute for the target address.

4.2.2.5 M/S_wrOrdered (Ordered Transfer)

This signal is driven by the master to indicate that the slave may not allow any subsequent requests (reads or writes) to get in between or ahead of the ordered write request. See the PLB specification for details on the associated PLB_ordered signal.

- Valid any time M/S_xxReq is asserted.

4.2.2.6 M_rdPriority, M_wrPriority (Request Priority)

This signal indicates the priority that the PLB arbiter should give to the associated request. "00" has the lowest priority, and "11" has the highest, although the exact priority given to the IP varies depending on the PLB arbiter that is used. For simplicity the user may tie this signal to a fixed value, or drive varying values depending on the importance of individual transfers.

- Must be valid any time M_xxReq is asserted.

4.2.2.7 M/S_rdType, M/S_wrType (Transfer Type)

This signal is asserted by the master to specify the type of transfer to be performed, as shown in Table 7. See the PLB specification on the PLB_type signal for details.

M/S_xxType (2:0)	*Definition*
000	Memory transfer.
001	DMA flyby transfer.
010	DMA buffered external peripheral transfer.
011	DMA buffered OPB peripheral transfer.
100	PLB slave buffered memory to memory transfer.
101	PLB slave buffered peripheral to/from memory transfer.
110	DMA buffered memory transfer.
111	PLB slave buffered memory to memory transfer with sideband signals.

Table 7: PLB Transfer Types

- Valid any time M/S_xxReq is valid.

4.2.3 PLB and Transfer Status

4.2.3.1 M_rdAbort, M_wrAbort (Master Transfer Abort)

The master may assert this signal to abort the current transfer, or to withdraw a transfer when the target slave requests rearbitration. The current transfer during pipelined operations always refers to the transfer that will be completed when the M_xxComp signal next goes high. Once accepted by the IPIF, a request can therefore not be aborted until all previous transfers have completed.

Since any data already on the bus when the abort signal is asserted must reach its destination, aborting a read transfer will result in up to two additional dwords of data arriving at the master in the cycles following the abort. No additional data must be provided by the master after aborting a write. Any transfer made in the clock cycle when the abort signal is asserted must also be considered. The transfer is not considered complete until the IPIF has asserted the complete signal (S_xxComp) for that transfer.

See page 22 on the master rearbitration signals for details about aborting a transfer in response to a rearbitration request made by the slave.

- May be asserted by the master any time a transfer is in progress. Should be negated otherwise.

4.2.3.2 S_rdAbort, S_wrAbort (Slave Transfer Abort)

This signal may be used by the slave to abort the current transfer. The corresponding terminate signal will be sent to the master by the IPIF, which will then end the transfer. Since any data already on the bus when the abort signal is asserted must reach its destination, aborting a write transfer will result in up to two additional dwords of data arriving at the slave in the cycles following the abort. No additional data is required after an aborted read from the slave. The request being made while the abort signal is asserted must also be acknowledged regardless of transfer direction. In all cases the transfer is not considered complete until the IPIF has asserted the complete signal (S_xxComp), and any requests arriving at the slave before this must be acknowledged.

- May be asserted by slave any time S_xxReq is active, negate otherwise.

4.2.3.3 S_rdBusy, S_wrBusy (Slave Busy)

The user IP slave may assert this signal to indicate that all further requests made to the LIS-IPIF in the associated transfer direction should be rearbitrated. See page 22 on the master rearbitrate signals for details.

- May be asserted by slave IP at any time.

4.2.3.4 M_rdError, M_wrError, M_Error (Master Error Indication)

These signals are provided to the user IP master to indicate that an error has occurred in the current transfer. M_xxError is asserted along with M_xxComp whenever a request is aborted or times out. M_Error is asserted when a slave explicitly asserts the error signal associated with that master, and may occur at any time during a transfer without causing an abort. Depending on the implementation of the slave, M_Error may or may not be asserted along with one of the other error signals.

- M_xxError: Asserted when the associated transfer is aborted or times out.
- M_Error: Asserted when an error is indicated explicitly by a slave.

4.2.3.5 S_rdError, S_wrError (Slave Error Trigger)

This signal may be asserted by the slave to signal an error to the master associated with the corresponding request (read or write). Asserting this signal will not abort the transfer, unless the master chooses to as a result of the error. Note that due to the split-bus scheme of the PLB, the master may not be able to determine which slave is asserting the error signal.

- May be asserted by slave any time S_xxReq is high.

4.2.3.6 M_Lock (Master Bus Lock)

By asserting this signal when making a request, the user IP master indicates that all further requests made by that master should occur atomically on the PLB, until the lock signal is negated for a full clock cycle. The LIS-IPIF master attachment locks the PLB in both directions (read and write) when the requested transfer is acknowledged by the slave, and maintains the lock even if the user IP master negates its request signal(s). Care should be taken that the lock duration is minimized, since all other PLB masters will be starved while the bus is locked.

- Asserted along with M_xxReq, and held high until the lock is released.

4.2.3.7 S_Locked (Bus Locked)

This signal indicates to the slave that the PLB is currently locked to a specific master. For details, see the description of the M_Lock signal above.

- Valid any time S_rdReq or S_wrReq are asserted.

4.2.3.8 M_rdRearbitrate, M_wrRearbitrate (Rearbitration Request)

This signal is passed to the master IP to indicate that the target slave of the current transfer is busy and has requested rearbitration of the transfer. If no actions are taken by the user IP in response to the rearbitration request, the LIS-IPIF will continue requesting that transfer until it is accepted by the slave.

To prevent the LIS-IPIF from repeating the same request indefinitely, the user IP master may assert the abort signal in response to the corresponding rearbitration request. The transfer requesting rearbitration will then be flushed from the IPIF, and the user IP must repeat the request at a later time (should it remain necessary). This allows the master to make additional transfers to other slaves while the original slave is busy.

- Asserted for one cycle each time the target slave requests rearbitration.

4.3 LIS-IPIF Configuration Parameters

Both the master and slave attachments of the LIS-IPIF provide numerous parameters that may be used to configure their operation, as shown in Tables 8 and 9. These options are available as generics to the top level, and may be used either to inform the IPIF about the overall system configuration, or to enable optional components of the IPIF. By enabling only those extensions required by the IP, the area overhead of the LIS-IPIF can be minimized.

<table>
<tr><th>Generic*</th><th>Type</th><th>Description</th></tr>
<tr><td>C_NUM_WIDTH</td><td>integer
(5)</td><td>Width of the M_xxNum signal used to specify the length of a transfer. This value should be at least 5 to allow for full 16-dword known-length bursts, or it may be larger to enable variable length bursts of arbitrary length.</td></tr>
<tr><td>C_EN_RECALC_ADDR</td><td>boolean
(false)</td><td>When true, the LIS-IPIF will recalculate the low-order bits of the target address to correspond with the byte enables provided by the master IP. This is required for compatibility with the PLB spec.
If the address furnished by the IP master is already correct, for example if the IP only requests data transfers with all bytes enabled, this option may be disabled to reduce the LIS-IPIF's area overhead.</td></tr>
<tr><td>C_ARBITRATION</td><td>integer
(0)</td><td>Sets the arbitration scheme to be used when the IP master requests both a read and write transfer simultaneously. If only a single request is being made when the LIS-IPIF is ready to accept a new transfer, that request will immediately be passed to the PLB regardless of arbitration scheme.
<table>
<tr><th>Value</th><th>Arbitration</th></tr>
<tr><td>0</td><td>Round-Robin</td></tr>
<tr><td>1</td><td>Read Priority</td></tr>
<tr><td>2</td><td>Write Priority</td></tr>
</table></td></tr>
<tr><td>C_EN_PIPELINING</td><td>boolean
(true)</td><td>When true, pipelining is enabled within the LIS-IPIF master, meaning that additional requests made by the IP may be accepted before the previous transfer has completed. This also means that in order to avoid duplicate transfers, the IP master must negate its request signal in the clock cycle after that request has been accepted. If pipelining is disabled, the request may be held high until the cycle after the complete signal is asserted.</td></tr>
</table>

* Generic names in italics are not yet fully implemented, and must assume their default value.

Generic	*Type*	*Description*
C_EN_FAST_ABORT	boolean (false)	When enabled, the abort signal provided by the IP will be forwarded to the PLB within the same clock cycle. This reduces the maximum number of additional transfers during a read operation to one, but opens a combinatorial path between IP and PLB.
C_EN_SRL16	boolean (true)	This option enables the use of LUT-based shift registers within the LIS-IPIF, greatly reducing the area overhead associated with data buffering. LUT-based storage may be undesirable in certain applications (e.g. for partial reconfiguration), in which case the use of SRL16s should be disabled.

Table 8: Master Configuration Constants

Generic[*]	*Type*	*Description*
C_BASEADDR	std_logic_vector	These 32-bit vectors specify the range of addresses to which the slave should respond. This range must be aligned to a positive power of two, i.e. the high address should be identical to the base address except for the requisite number of low-order bits. These bits should be negated in the base address and asserted in the high address.
C_HIGHADDR	std_logic_vector	
C_PLB_NUM_MASTERS	integer	Specifies the number of masters present on the PLB, therefore also the width of the Sl_MBusy and Sl_MErr signals.
C_PLB_MID_WIDTH	integer	Specifies the width of the PLB_masterID and S_xxMaster signals. Equivalent to LOG2(C_PLB_NUM_MASTERS).
C_EN_INC_ADDR	boolean (false)	When *true*, the target address presented to the IP will be incremented with each item of data that is transferred. Otherwise the address will refer to the first item of data for the entire transfer, and address incrementation is left to the IP.
C_ENFORCE_RANGE	boolean (false)	When enabled, the LIS-IPIF ensures that no overflow into addresses beyond the specified range can occur during burst transfers. If necessary the burst will be aborted by the LIS-IPIF, in the same manner as would occur if the IP asserts the S_xxAbort signal. This option may only be used in conjunction with C_EN_INC_ADDR.

* Generic names in italics are not yet fully implemented, and must assume their default value.

Generic	*Type*	*Description*
C_WATCHDOG	integer (0)	Enabling the transfer watchdog prevents deadlocks that may occur if the IP slave does not respond to a request. The LIS-IPIF will generate an abort and complete all handshaking for the current transfer when the S_xxReq signal has been asserted for the specified number of cycles without the S_xxAck signal being asserted by the IP. Set C_WATCHDOG to zero to disable the watchdog timer.
C_EN_FAST_ABORT	boolean (false)	When enabled, the abort signal provided by the IP will be forwarded to the PLB within the same clock cycle. This reduces the maximum number of additional transfers during a write operation to one, but opens a combinatorial path between IP and PLB.
C_EN_SRL16	boolean (true)	This option enables the use of LUT-based shift registers within the LIS-IPIF, greatly reducing the area overhead associated with data buffering. LUT-based storage may be undesirable in certain applications (e.g. for partial reconfiguration), in which case the use of SRL16s should be disabled.

Table 9: Slave Configuration Constants

5 Timing Examples

5.1 Write Burst Transfer

The signal flow for a typical write transfer from a LIS-IPIF master to slave is depicted in Figure 4. The transfer is initiated by the user IP asserting a master request along with all transfer qualifiers and the first item of data. These are immediately accepted by the IPIF and passed onto the PLB in the following clock cycle. Two cycles are required for arbitration before the request is passed to the slave and immediately acknowledged. Each data acknowledge is passed back to the master, and the transfer finishes with the complete signal being asserted for both the master and slave.

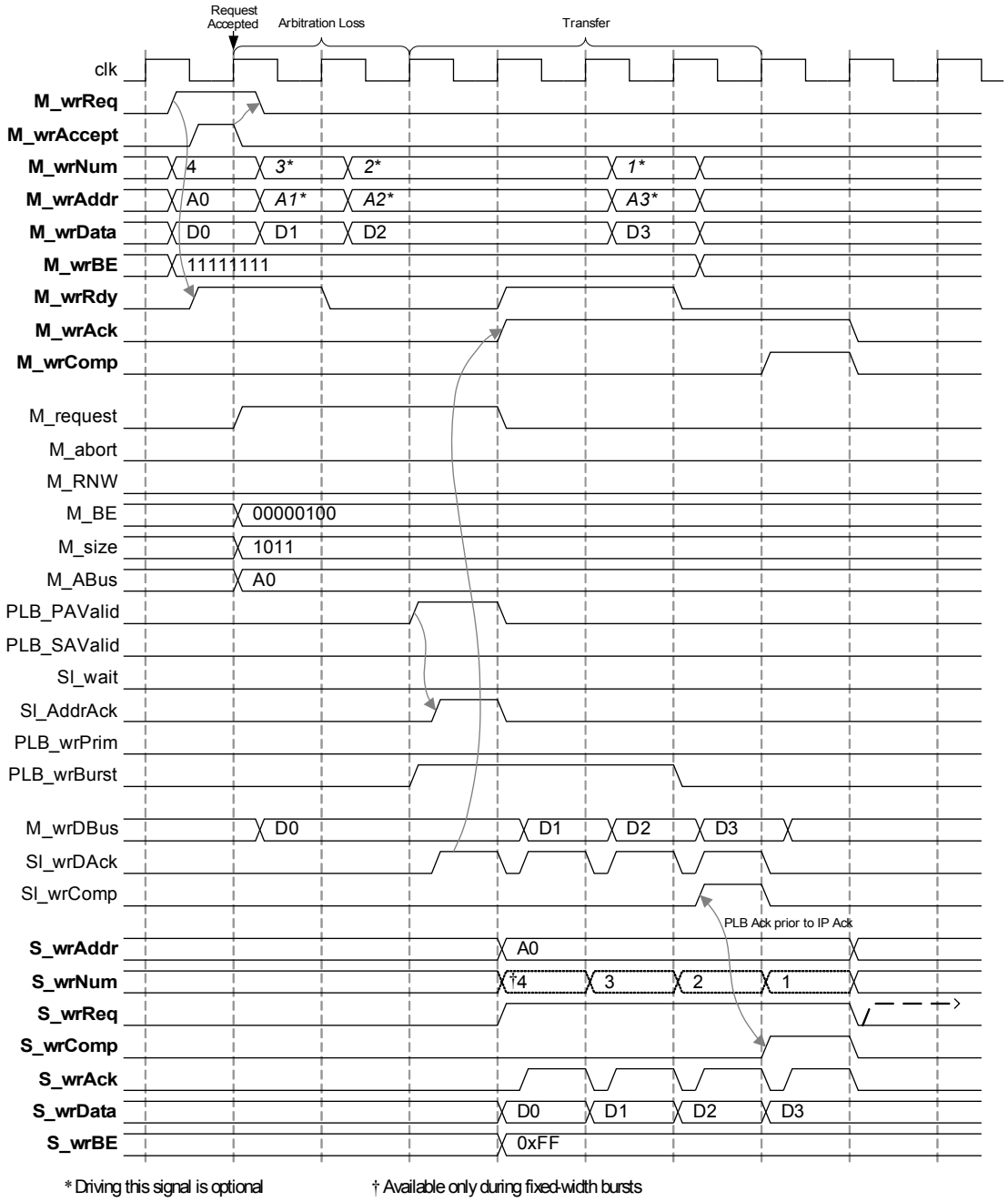


Figure 5: Write Burst Transfer

5.2 Read Burst Transfer

The read burst transfer shown in Figure 5 proceeds similarly to the write transfer demonstrated above, except for a reversal in data flow. The data provided by the slave IP is passed to the PLB one cycle after it is requested, and arrives at the master IP one clock cycle thereafter. A new request can be accepted by the LIS-IPIF in the cycle following the assertion of the complete signal.

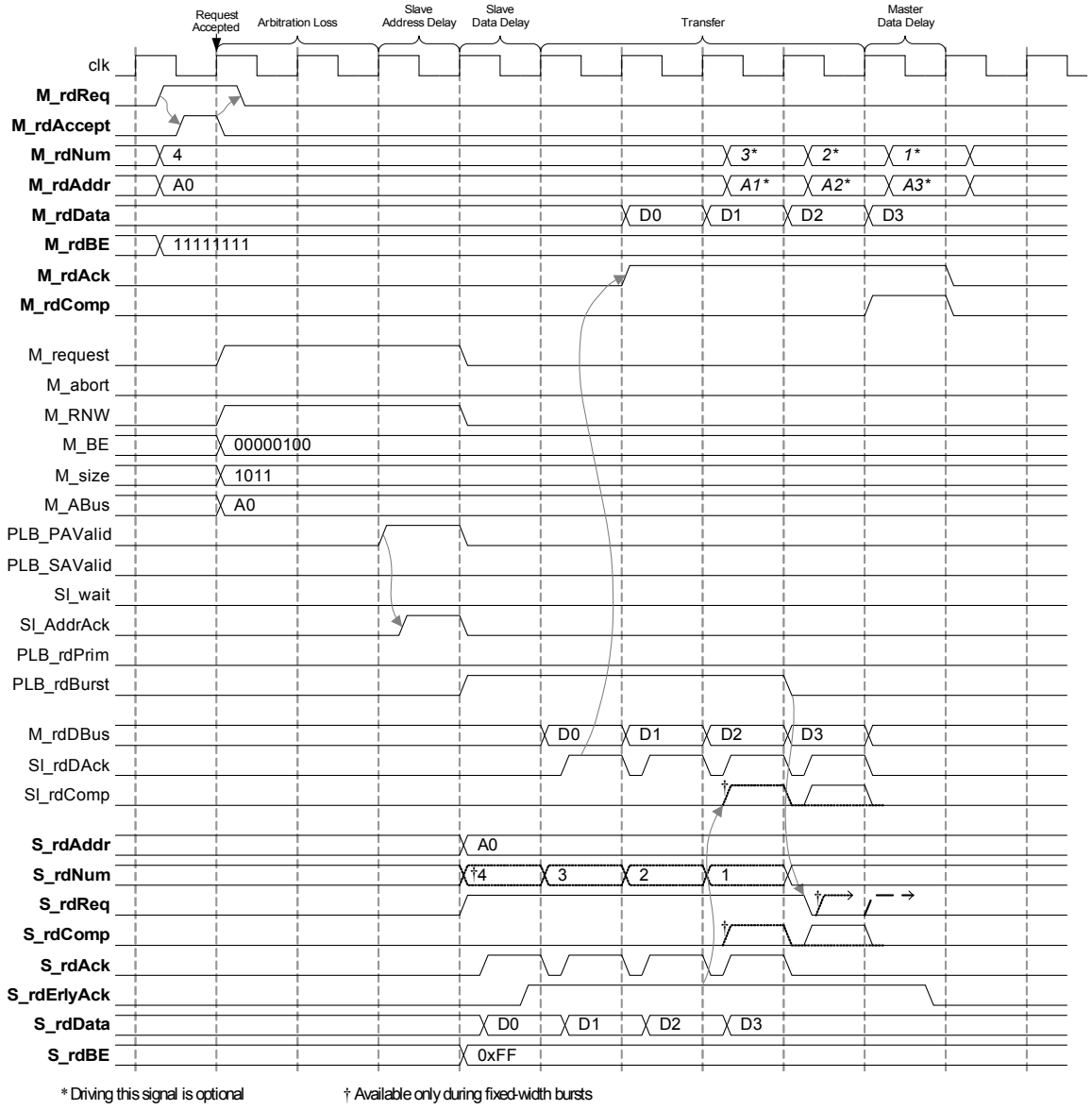


Figure 6: Read Burst Transfer

5.3 Write Transfer Abort

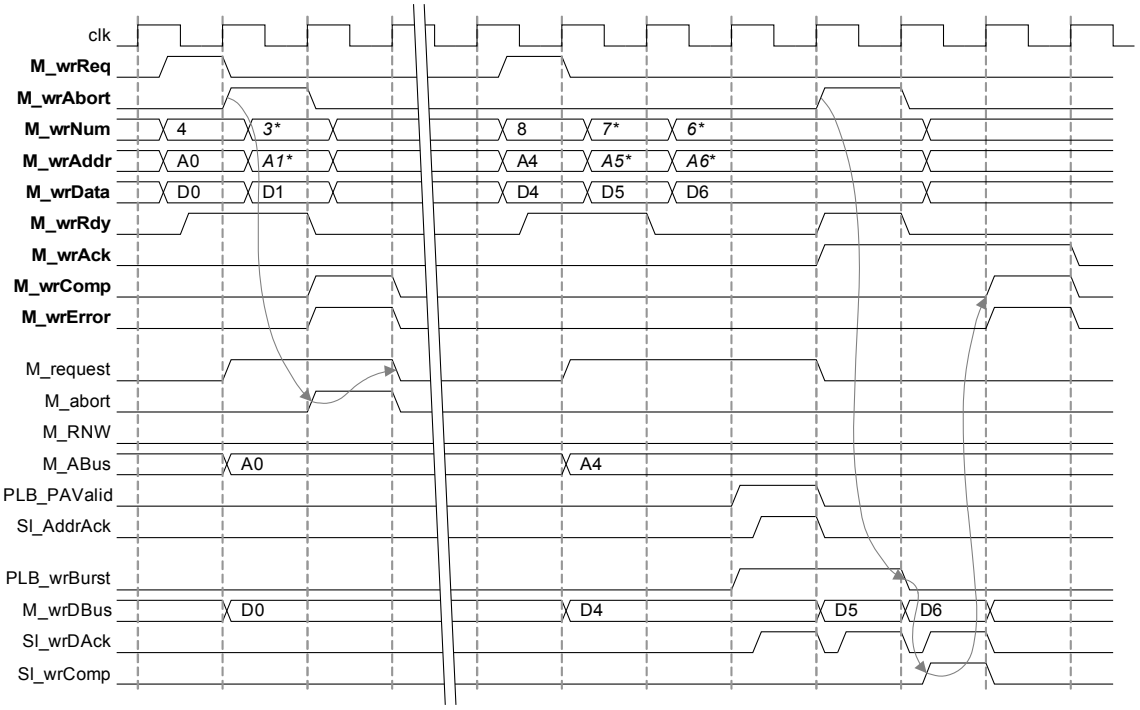


Figure 7: Write Transfer aborted by Master

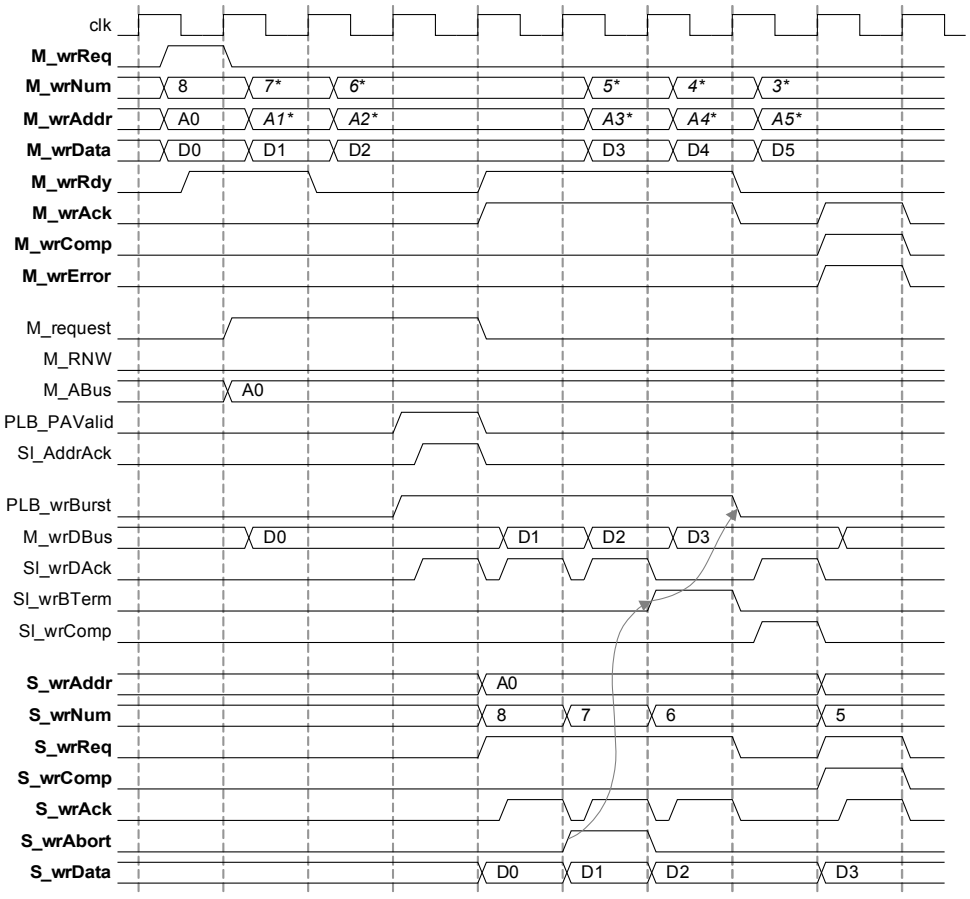


Figure 8: Write Transfer aborted by Slave

5.4 Read Transfer Abort

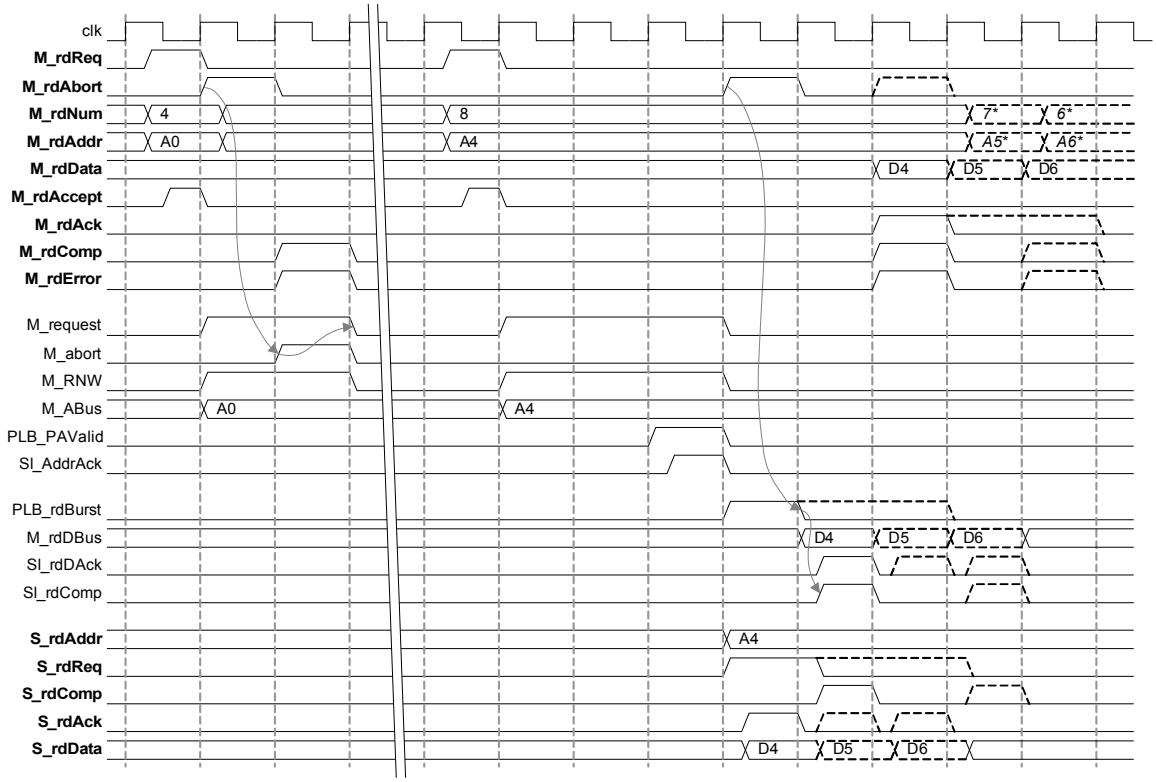


Figure 9: Read Transfer aborted by Master

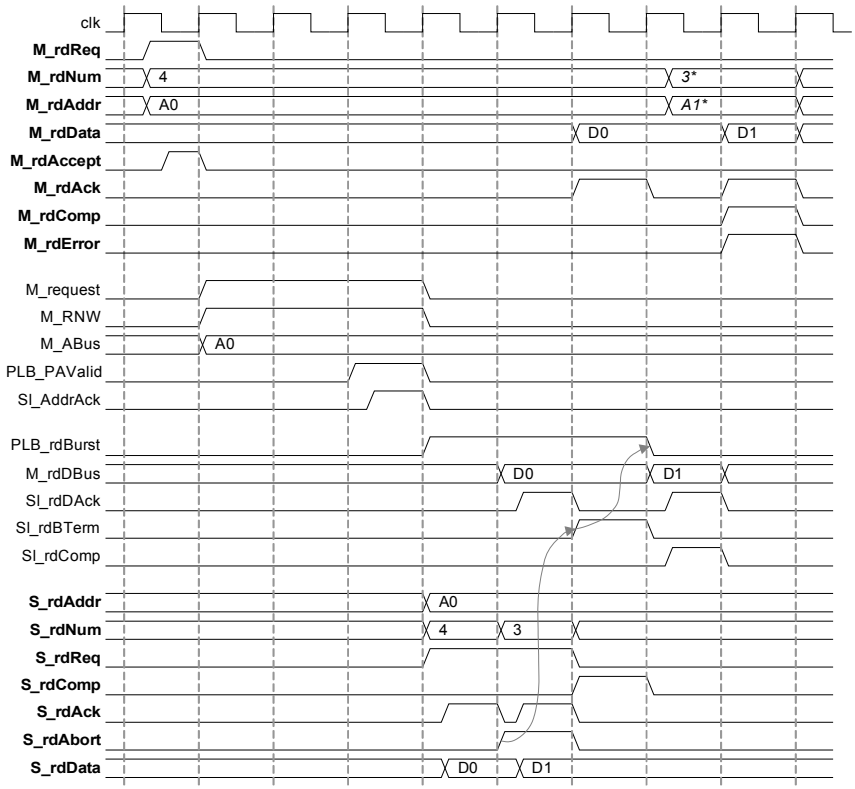


Figure 10: Read Transfer aborted by Slave

5.5 Pipelined Write Transfers

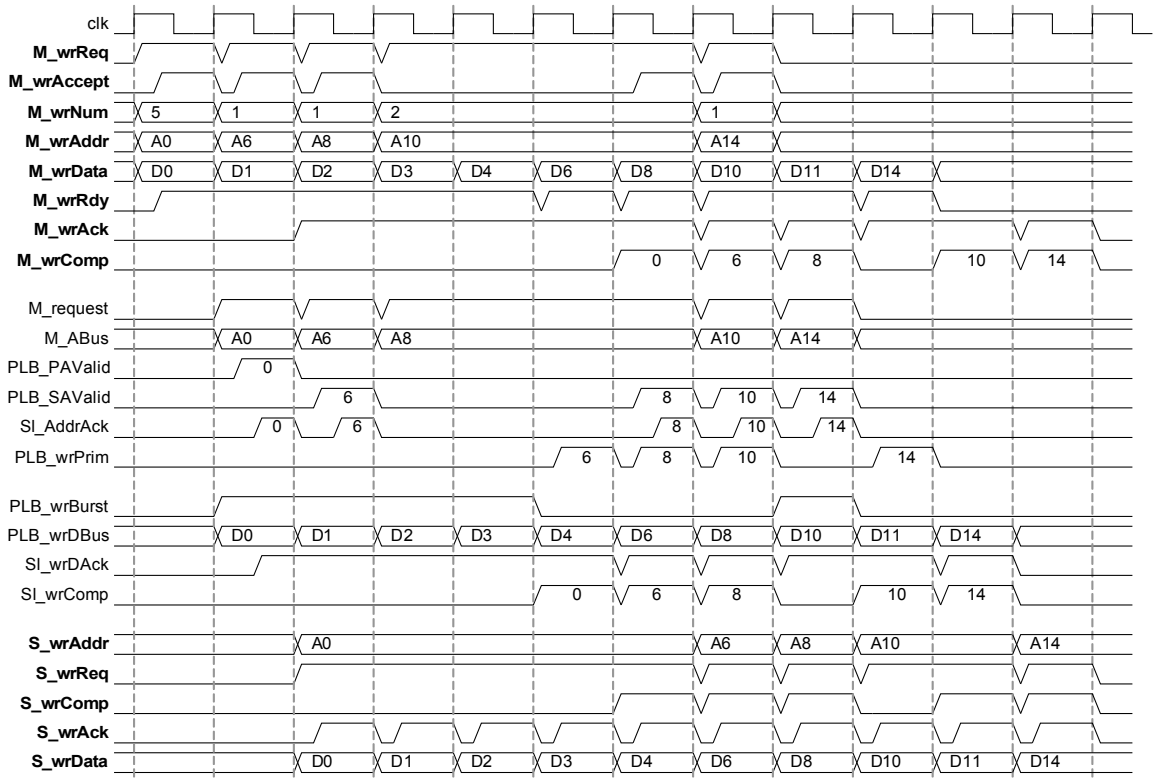


Figure 11: Pipelined Write Transfers (1-Cycle Arbitration)

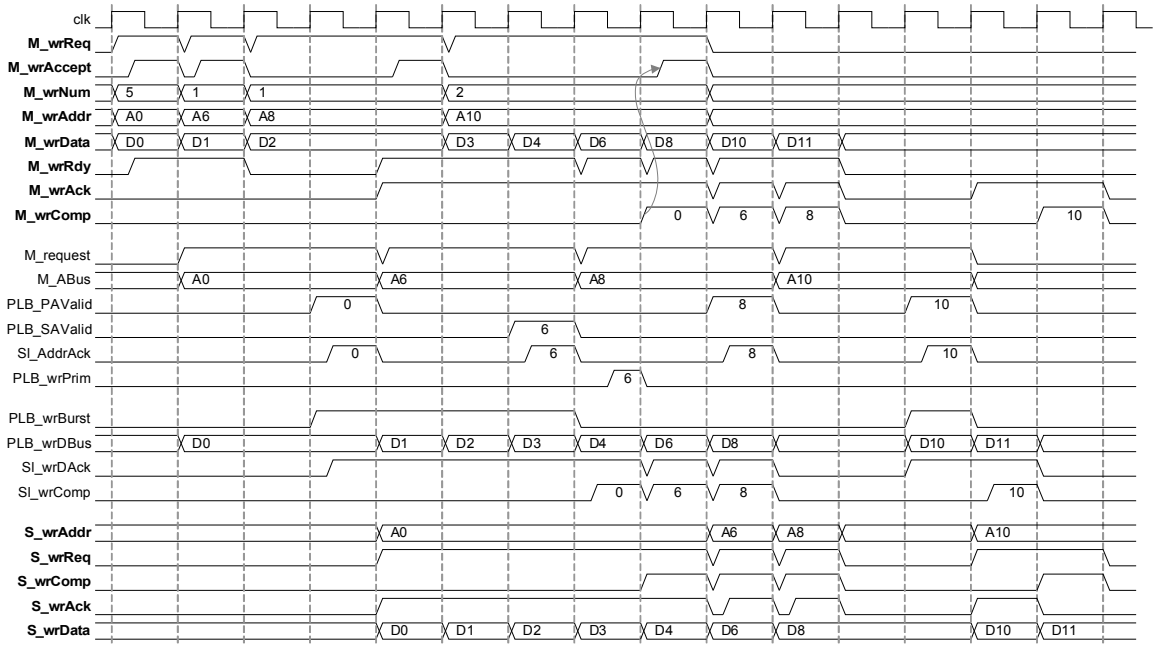


Figure 12: Pipelined Write Transfers (3-Cycle Arbitration)

5.6 Pipelined Read Transfers

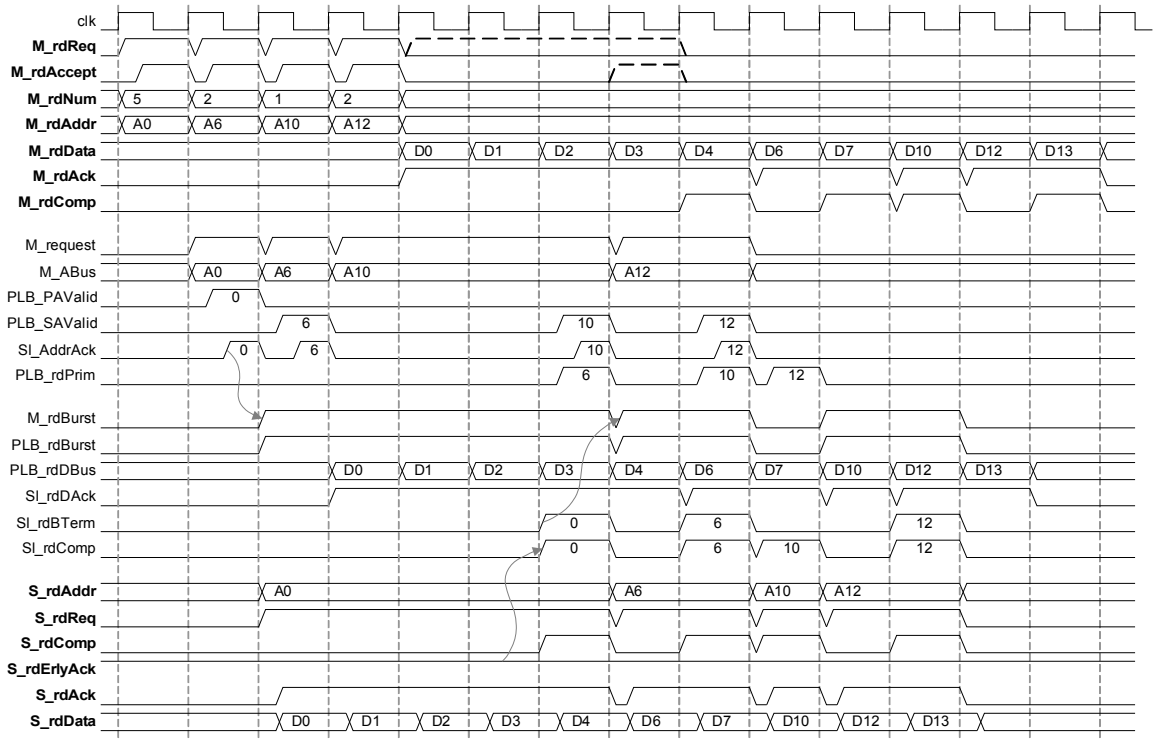


Figure 13: Pipelined Read Transfers (1-Cycle Arbitration)

Appendix A: Naming Conventions

All LIS-IPIC signals share a common naming convention composed of three parts. The first letter indicates whether the signal belongs to the Master or Slave attachment. This letter is followed by an underscore ('_') and the transfer direction associated with the signal (rd for read and wr for write, omitted if shared). The actual signal name comes last, each word beginning with a capital letter.