



PCI INTERFACE

D/TOS-ESM/ELV/157

E. Lama Vaquero
March 2000

PREFACE

This document is intended to be an appendix to complement the report written by Riccardo Locatelli about his work at ESTEC, this work being the implementation of a PCI Interface [1].

The names of some of the blocks have been changed to ease understanding. In such cases the names of the VHDL files they refer to, have been included in square brackets.

Special attention has been given to the connections, schematics and state diagrams to provide some further detail missing in the original document.

This note is intended to be read in conjunction with the original document written by Riccardo Locatelli, as the aspects already covered in that document have not been rewritten here.

CONTENTS

PREFACE.....	2
CONTENTS	3
1.- PCI INTERFACE. GENERAL OVERVIEW	4
2.- INTERFACE BLOCKS	5
2.1.- Synchronisation FIFO's	6
2.2.- PCI Core	7
2.2.1.- Target	8
Target FIFO Sequencer	9
Target State Machine.....	12
AD/CBE Buffers Target	14
Parity & Error Target.....	15
2.2.2.- Master.....	16
FIFO IN Sequencer.....	17
FIFO OUT Sequencer.....	20
Master State Machine	21
AD/CBE Buffers Master	23
Parity&Error Master	23
2.2.3.- Configuration block.....	25
3.- SIMULATIONS	26
3.1.- Write cycles.....	26
3.1.1.- Single Memory Write Cycle	27
3.1.2.- Burst Write Cycle.....	28
3.1.3.- I/O Write Cycle	29
3.2.- Read cycles.....	29
3.2.1.- Single Memory Read Cycle	29
3.2.2.- Burst Read Cycle.....	33
3.2.3.- I/O Read Cycle	35
3.3.- Configuration cycle	35
4.- CONCLUSIONS	37
REFERENCES.....	39

1.- PCI Interface. General overview

The PCI Local Bus is a high performance, 32- or 64-bit bus with multiplexed address and data lines. The bus is intended for use as an interconnect mechanism between highly integrated peripheral controller components, peripheral add-in boards, and processor/memory systems [2].

A simple PCI Local Bus architecture is shown in figure 1.

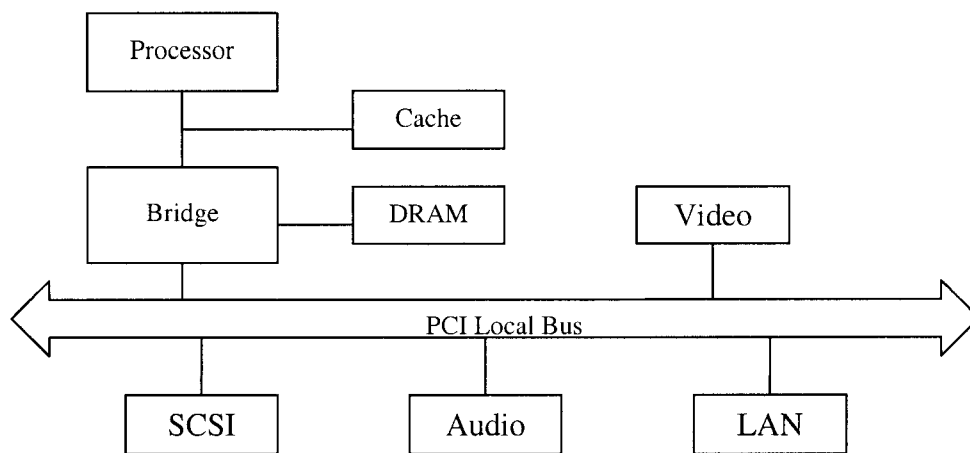


Figure 1. Example of a PCI System.

In this example, the processor/cache/memory subsystem is connected to PCI through a PCI bridge. Connected to the bus are typical PCI applications like LAN's, graphics adapters or SCSI cards. For an application to be connected to the PCI Local Bus, a PCI interface is needed. The application can be either a master, target or both.

A "master" is the entity that initiates the transaction on the PCI bus; it requests write or read accesses to one of the three address spaces of the PCI bus (configuration, I/O or memory). A "target" is the entity that responds to the transaction initiated by a master application.

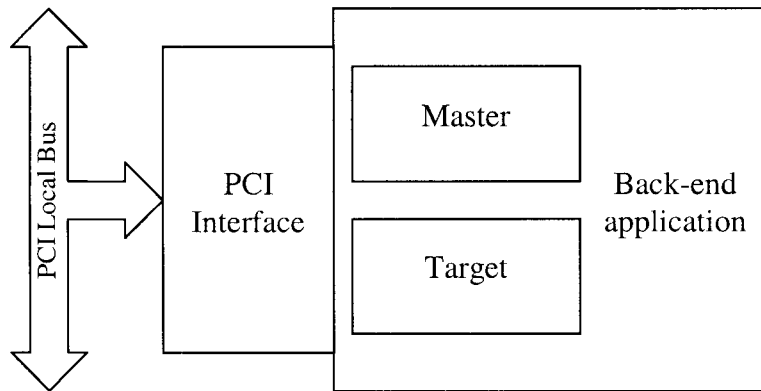


Figure 2. PCI Interface connecting a back-end application to a PCI Local Bus.

The PCI interface ensures that the information exchanges conform to the PCI specification by generating the appropriate signals.

The PCI bus specification defines the interaction between two PCI resources: the PCI bus master and PCI bus target. As with other bus specifications, the PCI specification defines a protocol for the PCI bus master to read and write data from and to a target, respectively. It also defines access cycles to a configuration address space to identify and establish hardware requirements of PCI resources. In addition, it defines a special cycle to broadcast a message to all PCI resources, and an interrupt acknowledge cycle¹ to return the interrupt vector to the platform CPU [3].

The PCI bus protocol is defined as synchronous. All of the signal lines and bus activity is referenced to the rising edge of a bus clock (CLK signal line) except for the SERR# (system error), RST# (RESET), and INT#¹ (interrupt) signal lines [3].

The PCI interface described in this document is an ASIC designed in VHDL. It implements a 32-bit, 33MHz PCI Master/Target Bus sequencer, compliant with the PCI Local Bus Standard, Revision 2.1.

2.- Interface blocks

The PCI Interface [LINK_TEST_PCI] connects to the PCI bus on one side and to a back-end application on the other.

The structure of the whole PCI Interface is shown in figure 3. The design is divided in two main parts: the PCI core and the synchronisation FIFO's. A top-down approach has been chosen to describe the blocks in the system.

¹ Only in the extended version (64-bit).

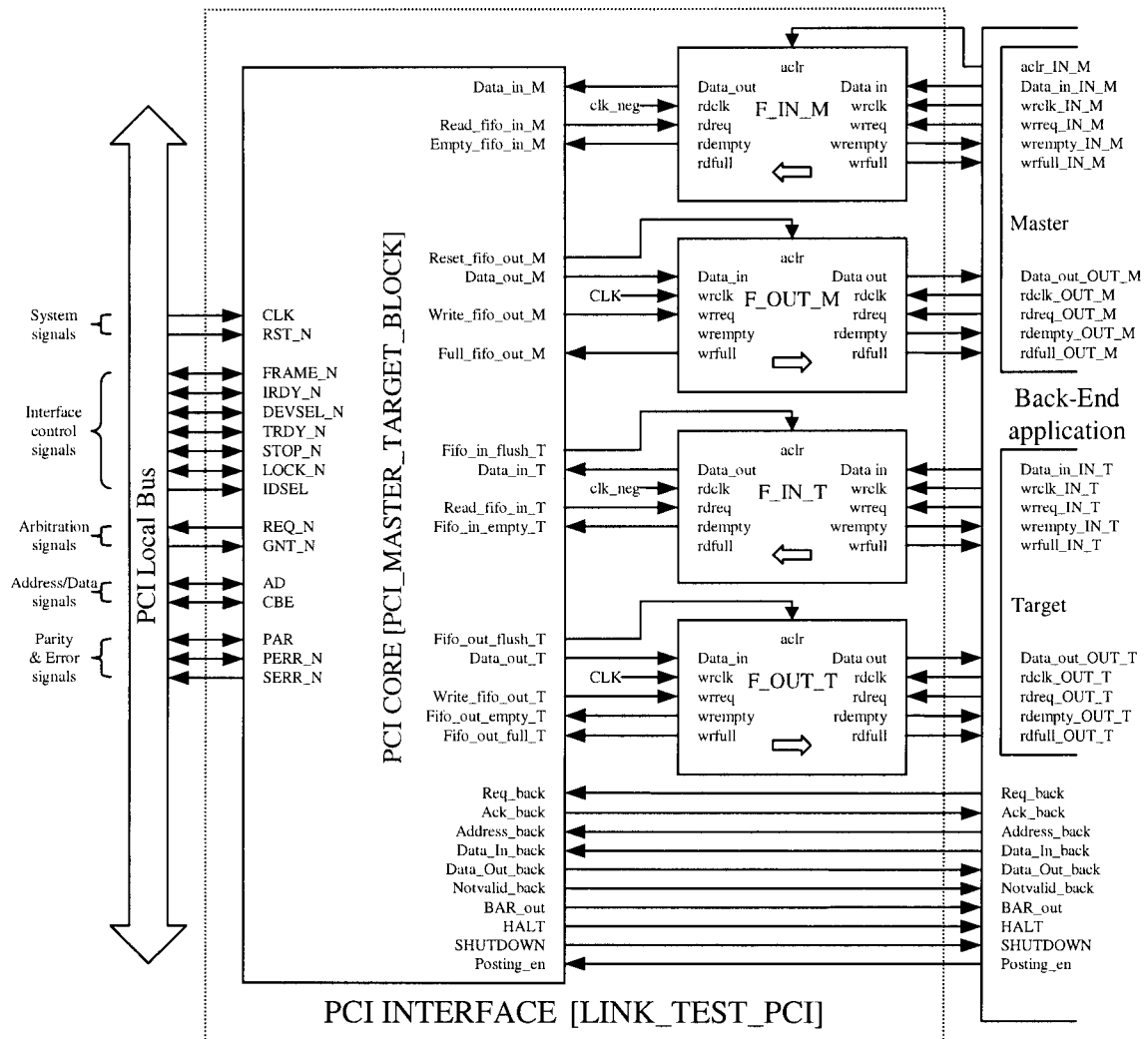


Figure 3. PCI Interface [LINK_TEST_PCI] connection.

2.1.- Synchronisation FIFO's

The FIFO's designed for the PCI interface were implemented at the gate level, thus requiring a large amount of memory in the simulation program to perform tests. For the simulations performed for this report, they have been substituted for behavioural models in VHDL.

The PCI Local Bus and the back-end applications run with different clock lines. These clocks may differ in frequency and/or phase. Four FIFO's have been added for synchronisation: F_IN_M, F_OUT_M, F_IN_T and F_OUT_T. These FIFO's use different clocks for read and write cycles. In that way, they serve as a data buffering between both systems.

The back-end application can work as master, target or both, for that reason, four FIFO's are needed. For the master side, input data is written synchronous with the back-end application clock and read synchronous with the PCI Local Bus clock, and output data is written synchronous with the PCI Local Bus clock and read

synchronous with the back-end application clock. The same structure is repeated for the target side.

2.2.- PCI Core

The three blocks that compose the PCI Core are the Master, the Target and the Configuration blocks (see figure 4).

The back-end application can be a Master device, a Target device or both. That means separate data paths must be provided so that Master and Target parts of the same application can operate independently.

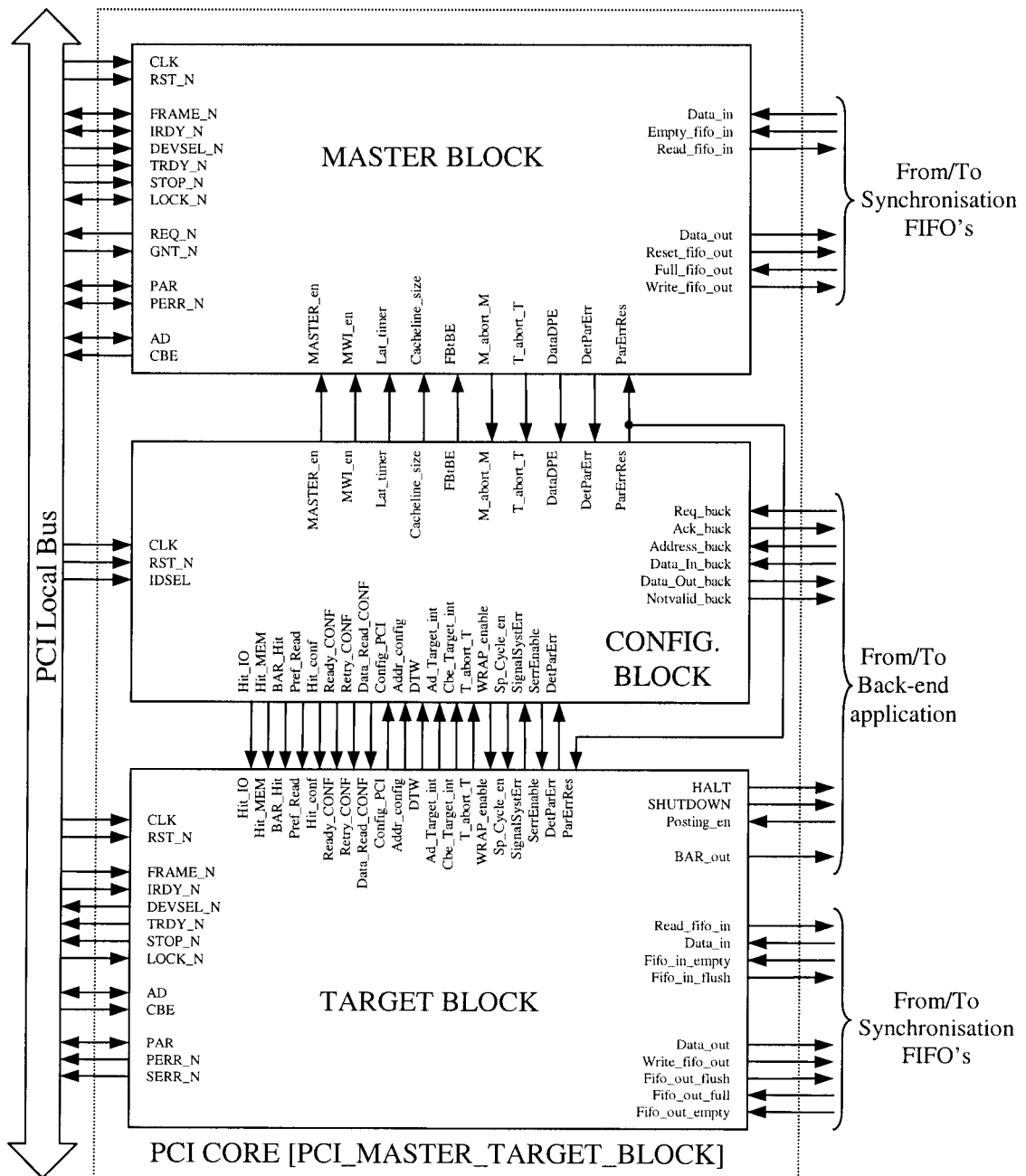


Figure 4. Internal structure of the PCI Core.

The Master Block in the PCI Core is intended to interface with the Master side of the back-end application (if any), and the Target Block in the PCI Core is designed to interface with the Target side of the back-end application (again, if any).

2.2.1.- Target

The target block receives/sends the data, though the corresponding synchronisation FIFO's, from/to the Target side of the back-end application.

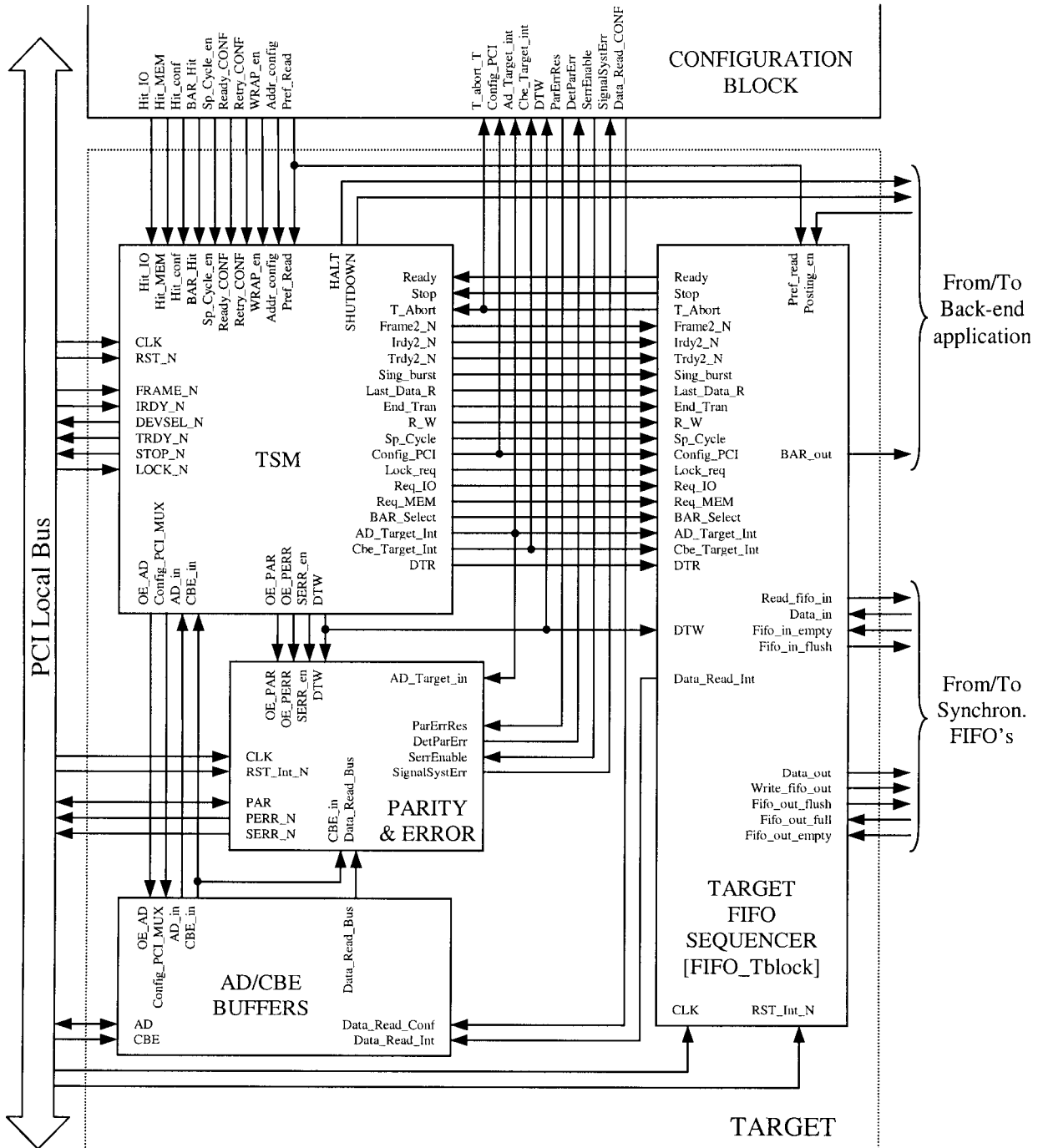


Figure 5. Blocks composing the target block in the PCI Core.

On command and address cycles, the Target Block receives the information from the PCI Local Bus and sets the parameters for the consequent cycle(s).

On write cycles, the Target Block receives the data from the PCI Local Bus and, after decoding it, is sent to the back-end application via the F_OUT_T synchronisation FIFO or to the configuration space.

On read cycles, data is received from the back-end application via the F_IN_T synchronisation FIFO or from the configuration space and written to the PCI Bus.

The sub-blocks composing this system are described in the following sections.

Target FIFO Sequencer

The Target FIFO Sequencer [FIFO_Tblock] serves as the link between the synchronisation FIFO's (F_IN_T and F_OUT_T) and the Target State Machine. It manages the transfer of data from and to the back-end application. The data arriving from the back-end application is grouped in 40-bit words. These forty bits are divided in C/BE# (4 bits), AD (32) and Req_int (4 bits).

The lowest bits received, Req_int[3..0], define the access type. It is encoded according to the table below.

Bit [3..0]	Description
1111	Address single
1110	Address burst
1101	Address burst prefetchable
1011	Address single + fast back-to-back
1010	Address burst + fast back-to-back
1001	Address burst pref. + fast back-to-back
0111	Address single + Lock request
0110	Address burst + lock request
0101	Address burst pref. + Lock request
0011	Addr. Single + Fast b-to-b + Lock req.
0010	Addr. Burst + Fast b-to-b + Lock req.
0001	Addr. Burst pref. + Fast b-to-b + Lock req.
0100	Data
1000	Last Data / Stop

A brief description of the terms in the table follows:

- Single is defined as one access of a memory resource.
- Burst is defined as multiple read and writes during one access cycle. The addressing is incremental relative to a base address.
- Prefetchable returns all the bytes in the given address independently of the C/BE# signal lines.
- Fast back-to-back access avoids the idle phases between access cycles when the target is being accessed by the same master

- Lock. The protocol by which a PCI bus master can only access a specific target and the specific target can only be accessed by the aforementioned PCI bus master. The locking of a memory access simply means that only one PCI bus master (called Lock master) can access the locked resource (target). Any PCI bus master can become the Lock master of a given target.
- Data phase: the portion of the PCI bus cycle when data and the byte enable information are present on the bus.
- Last data/Stop indicates that this is the last data in a transfer.

The 32 following bits, AD[31..0], indicate the address where the transaction has to take place, or the data to be transferred.

Bus commands indicate to the target the type of transaction the master is requesting. These bus commands are encoded on the C/BE[3..0]# lines during the address phase. PCI bus command encodings are listed and described below [2]:

C/BE#[3..0]	COMMAND TYPE	SUPPORTED
0000	<i>Interrupt Acknowledge</i>	No
0001	<i>Special Cycle</i>	Yes
0010	<i>I/O Read</i>	Yes
0011	<i>I/O Write</i>	Yes
0100	<i>Reserved</i>	---
0101	<i>Reserved</i>	---
0110	<i>Memory Read</i>	Yes
0111	<i>Memory Write</i>	Yes
1000	<i>Reserved</i>	---
1001	<i>Reserved</i>	---
1010	<i>Configuration Read</i>	Yes
1011	<i>Configuration Write</i>	Yes
1100	<i>Memory Read Multiple</i>	Yes
1101	<i>Dual Address Cycle</i>	No
1110	<i>Memory Read Line</i>	Yes
1111	<i>Memory Write & Invalidate</i>	Yes

- Special Cycle provides a simple message broadcast mechanism on PCI.
- I/O Read is used to read data from an agent¹ mapped in I/O Address Space. AD[31..0] provide a byte address. All 32 bits must be decoded. The byte enables indicate the size of the transfer and must be consistent with the address byte.
- I/O Write is used to write data to an agent mapped in I/O Address Space. All 32 bits must be decoded. The byte enables indicate the size of the transfer and must be consistent with the byte address.
- Memory Read is used to read data from an agent mapped in the Memory Address Space.
- Memory Write is used to write data to an agent mapped in the Memory Address Space.
- Configuration Read is used to read the Configuration Space of each agent.
- Configuration Write is used to transfer data to the Configuration Space of each agent.

¹ Agent: an entity that operates on a computer bus.

- Memory Read Multiple is identical to the Memory Read command except that it additionally indicates that the master may intend to fetch more than one cache line before disconnecting.
- Memory Read Line is identical to the Memory Read command except that it additionally indicates that the master intends to fetch a complete cache line.
- Memory Write and Invalidate is identical to the Memory Write command except that it additionally guarantees a minimum transfer of one complete cache line.

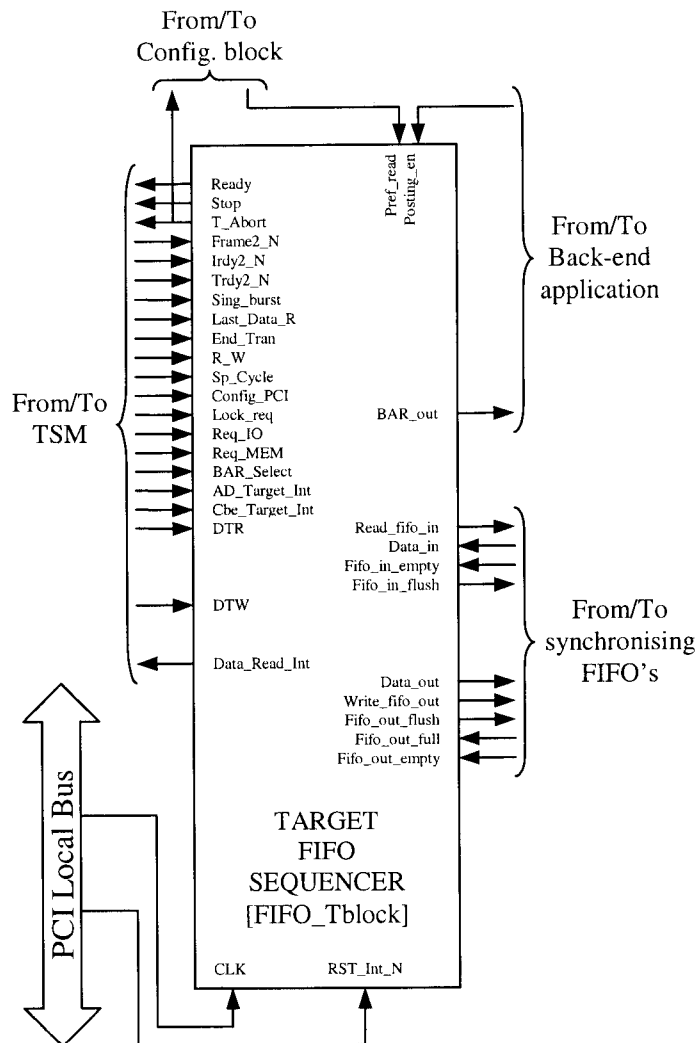


Figure 6. Target FIFO sequencer [FIFO_Tblock].

Figure 6 shows the interconnection lines between the Target FIFO Sequencer and the rest of the blocks in the Target PCI Core.

The main element in the Target FIFO Sequencer is the state machine shown in figure 7. The four states represent:

- Idle: when the target block is not involved in any transaction.
- Mem_W: when data is being written in the back-end application target.
- Mem_R: when data is being read from the back-end application target.
- Not_Mem: used for I/O and configuration operations.

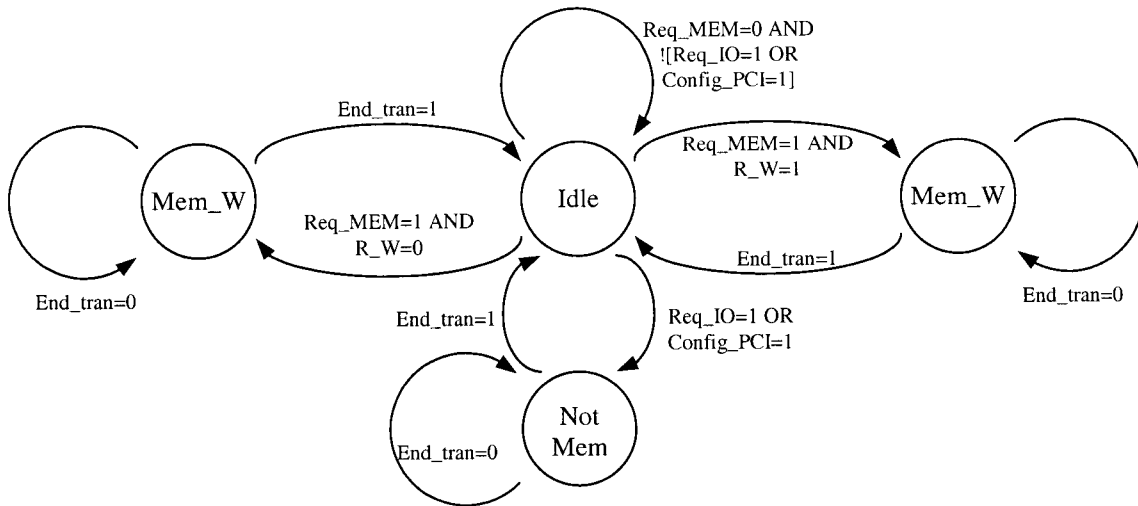


Figure 7. Target FIFO Sequencer [FIFO_Tblock] State Diagram.

All possible transactions are included in one of these states, where the appropriate signals are driven according to the tables shown earlier.

Target State Machine

This sub-block implements the main function in the Target block of the PCI Core. Once the cycle type is decoded, it produces the appropriate signals to command the rest of the sub-blocks when the back-end application becomes the slave of a bus transaction. Figure 6 shows the interconnections of the TSM block with the other blocks in the PCI Core Target.

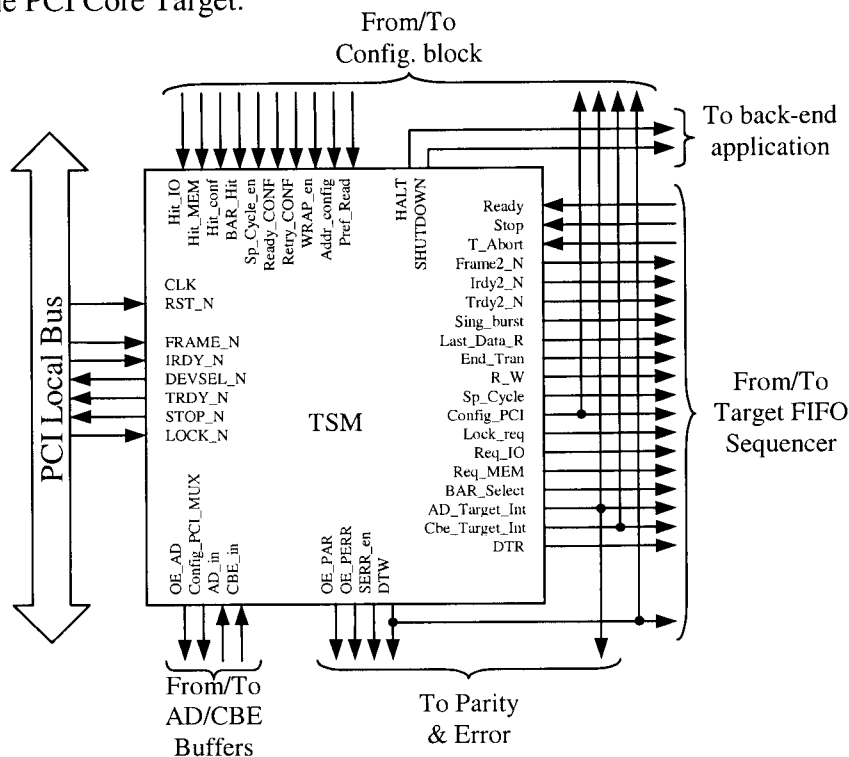


Figure 8. Target State Machine block.

The main part in the architecture of the TSM block is the two state diagrams shown in figures 9 and 10. A description of the states can be found in [1].

The main state diagram (figure 9) generates the signal lines that command the rest of the blocks following the PCI Specification.

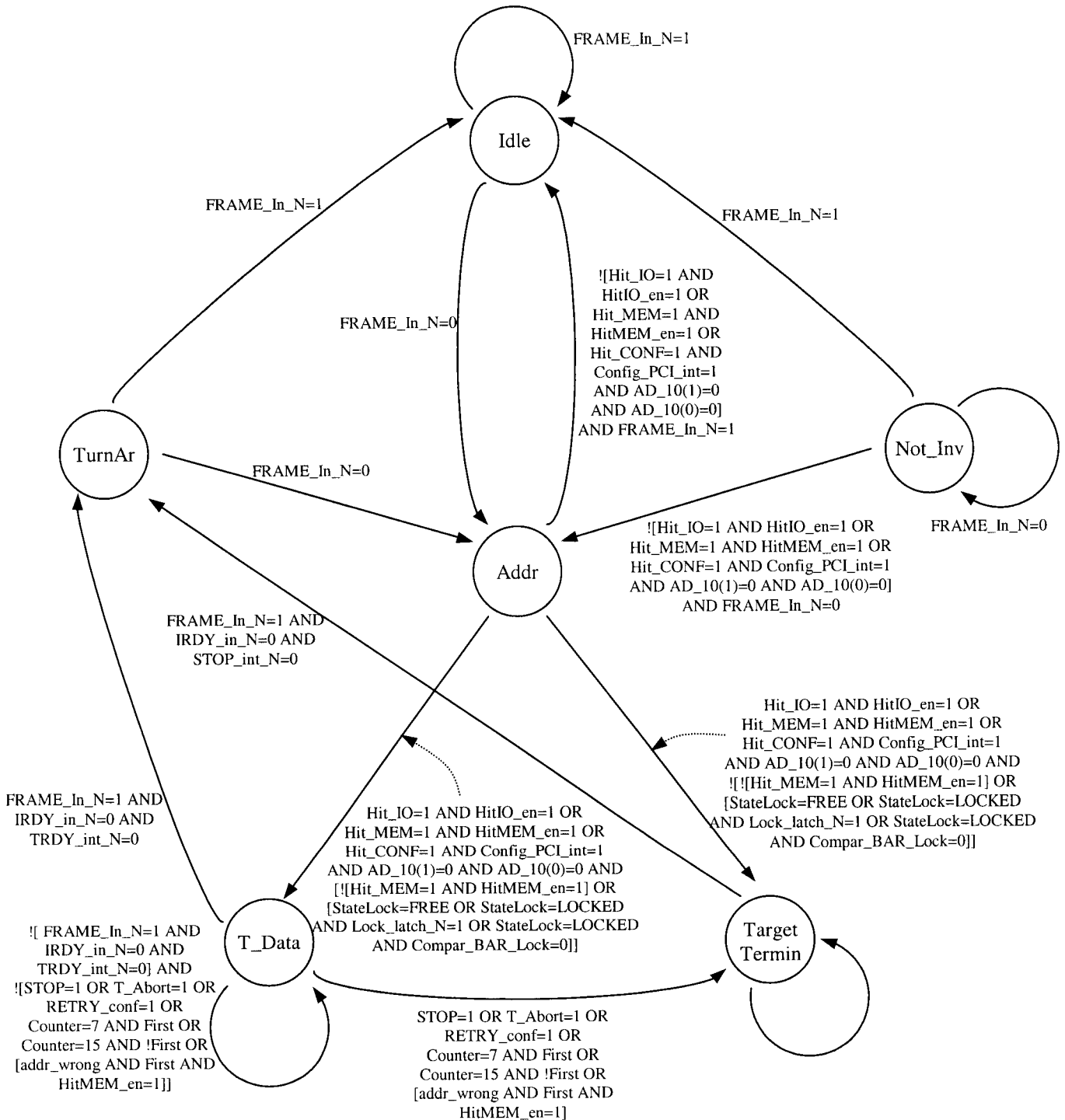


Figure 10. Main state diagram in the Target State Machine.

A Lock state machine is needed to prevent other masters accessing the target when a lock access is taking place.

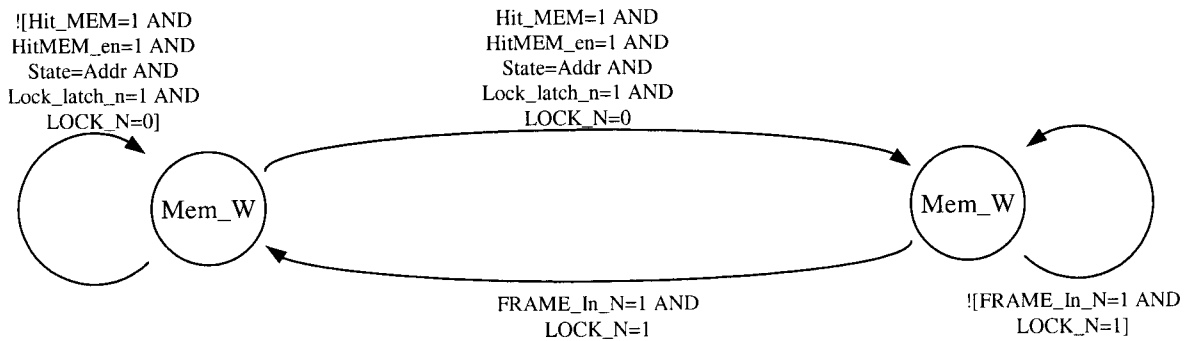


Figure 10. Lock State diagram.

AD/CBE Buffers Target

The AD/CBE buffers connect the 32-bit address/data and the 4-bit Command/Byte Enable lines of the PCI bus with the internal buses. There are two different data paths for the Master and the Target part of the PCI Core.

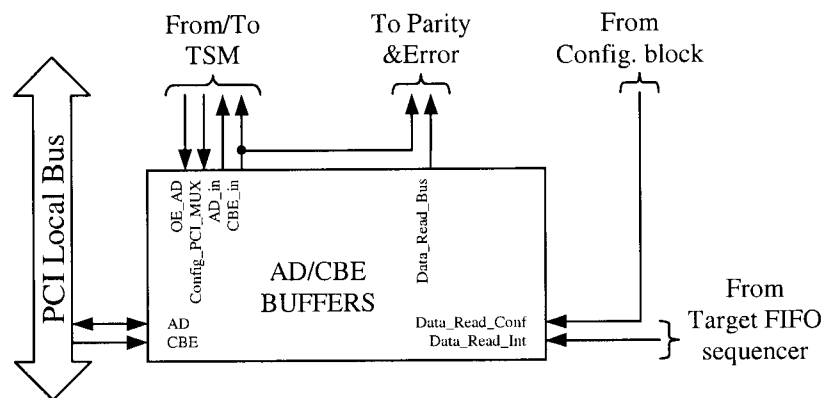


Figure 11. AD/CBE Buffers block.

The internal structure of the block is shown in figure 12. AD_In and CBE_In are provided to the TSM for sampling data during write operations. During read operations, data may arrive from the back-end application (Data_Read_in) or from the Configuration Space (Data_Read_conf). After selecting the correct line, it is transferred to the PCI bus through three-state ports controlled by TSM with Output Enable signals.

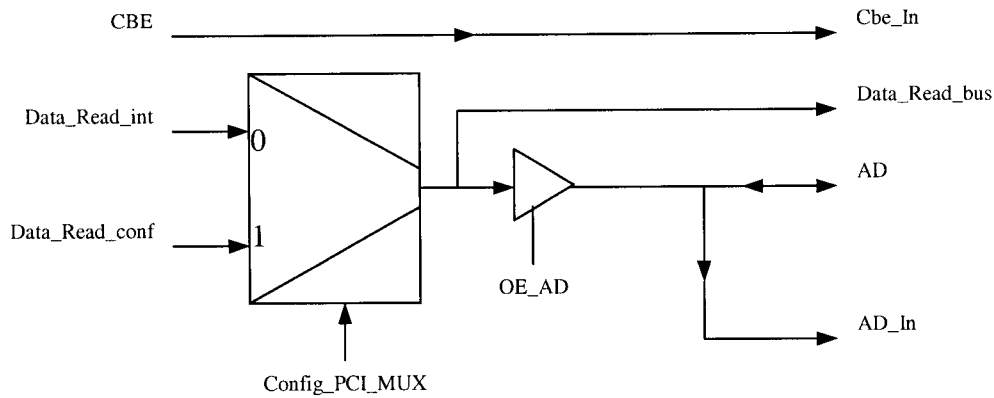


Figure 12. Schematics in the AD/CBE Buffers Block.

Parity & Error Target

The Parity & Error Target block computes parity and reports errors when the PCI Bus Sequencer is working as Target.

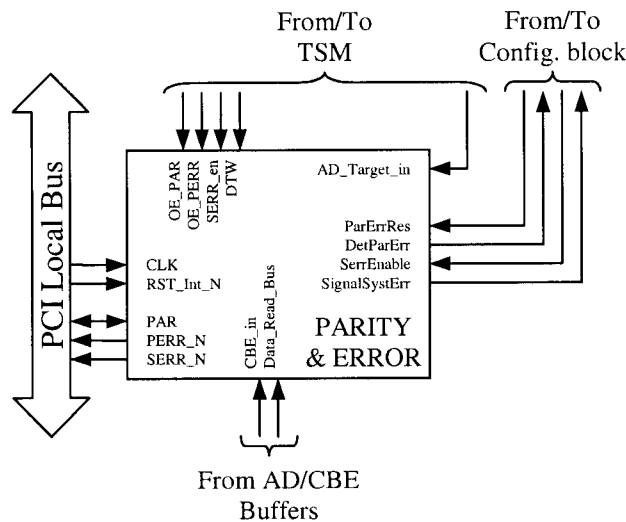


Figure 13. Parity & Error Target Block.

During the Data phase of read operation, this block generates the Parity signal. Otherwise it checks the incoming Parity signal. The PERR# signal is driven during Data phases and SERR# during Address phases and Special cycles.

Figure 14 shows the schematics of the Parity & Error block. Underlined are the input/output lines, the rest being internal signals.

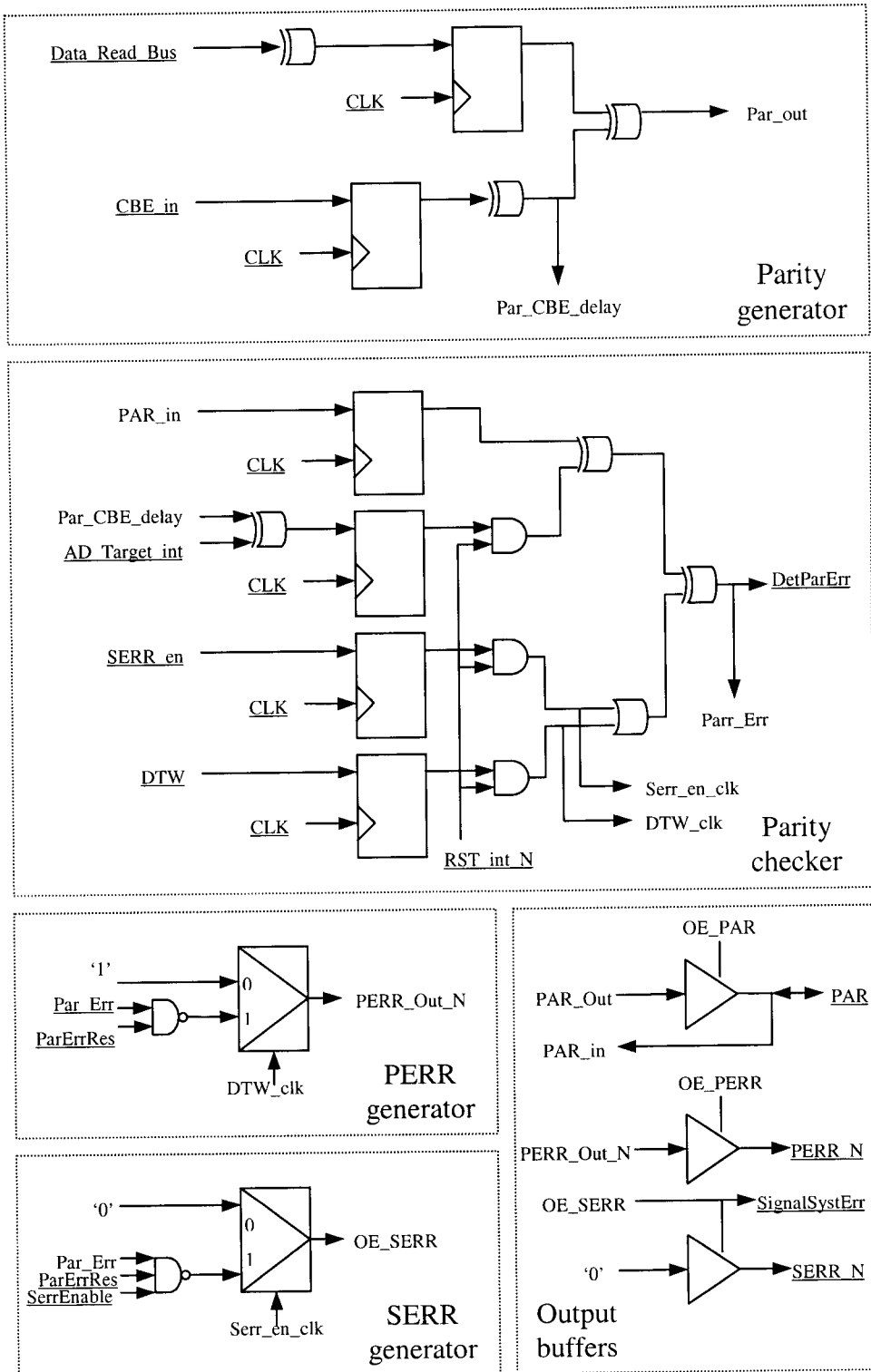


Figure 14. Schematics in the Parity & Error Target Block.

2.2.2.- Master

The master block receives/sends the data, though the corresponding synchronisation FIFO's, from/to the Master side of the back-end application. The sub-blocks composing this system are described in the following sections.

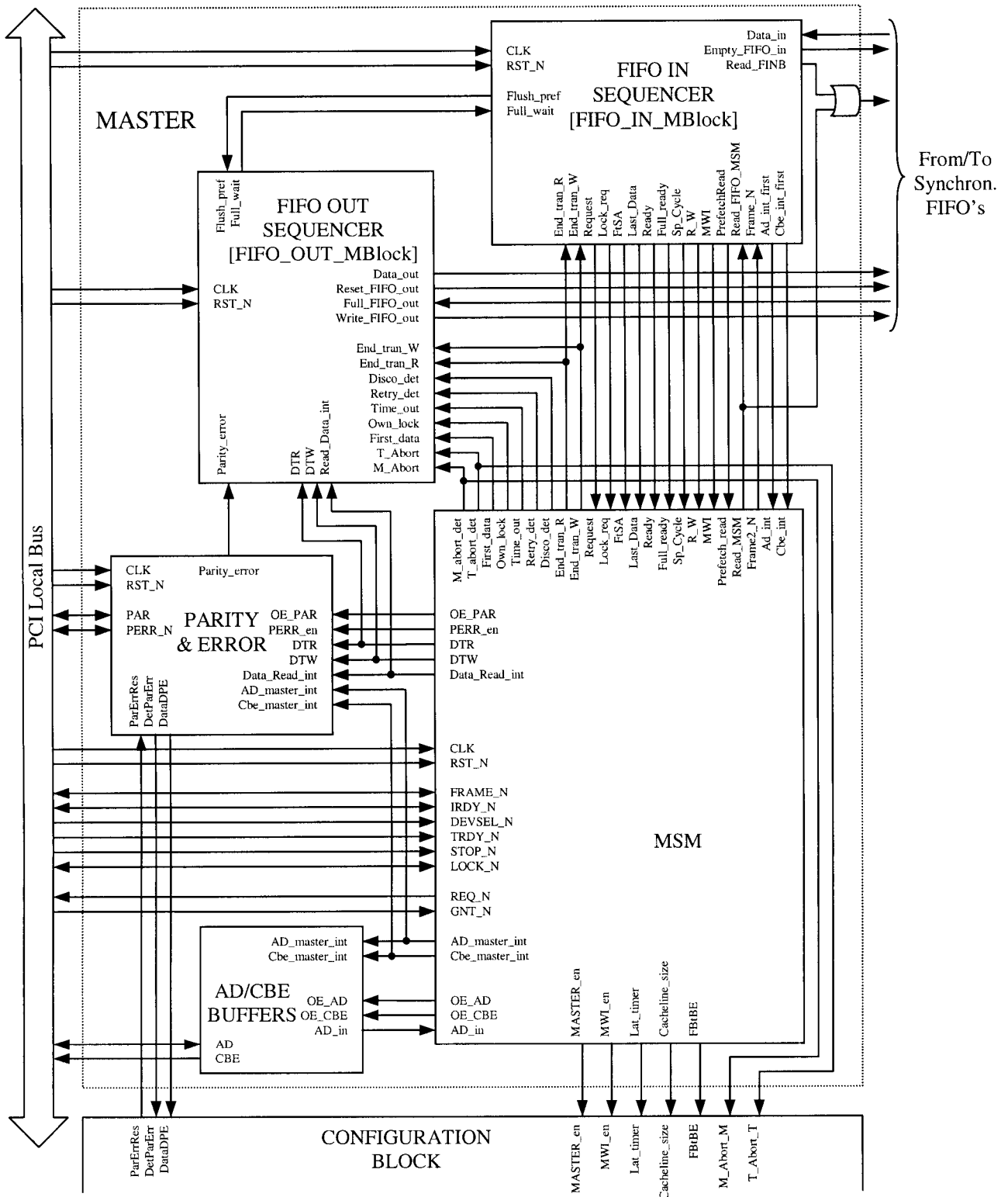


Figure 15. Blocks composing the master block in the PCI Core.

FIFO IN Sequencer

The Master FIFO IN Sequencer [FIFO_IN_Mblock] serves as the link between the input synchronisation FIFO (F_IN_M) and the Master State Machine. It manages the

The rest of the schematics in this block are shown in figure 18. The data arriving from the back-end application is grouped in 40-bit words. These forty bits are divided in C/BE# (4 bits), AD (32) and Req_int (4 bits). The decoders handle this information and set the different signal lines according to the tables in the *Target FIFO Sequencer* section.

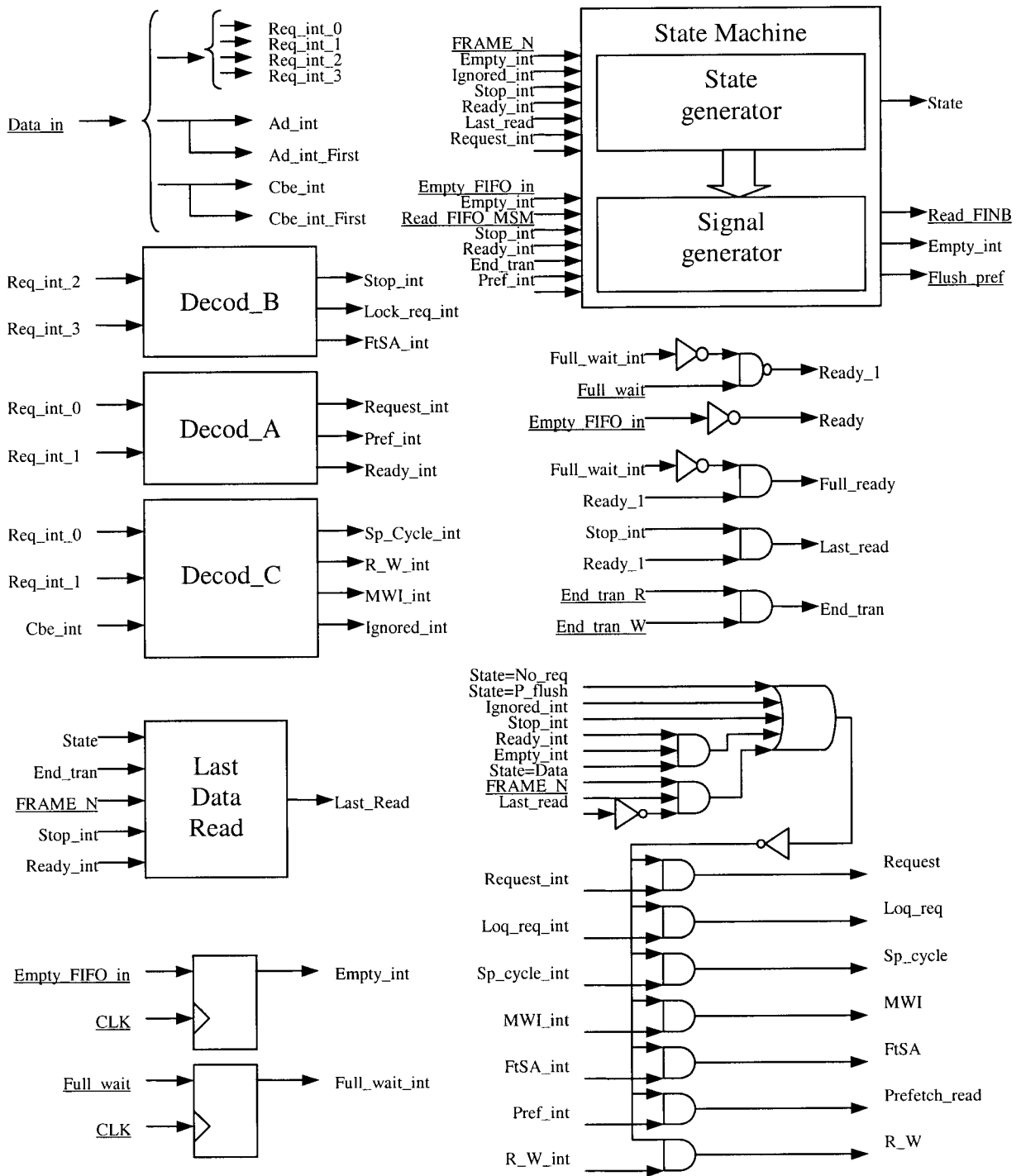


Figure 18. Schematics of the FIFO IN Sequencer Block [FIFO_IN_Mblock].

FIFO OUT Sequencer

This block [FIFO_OUT_Mblock], as shown in figure 19, manages the handshake between the Master State Machine and the back-end application, handling in the correct way the information transferred to the output synchronisation FIFO [F_OUT_M].

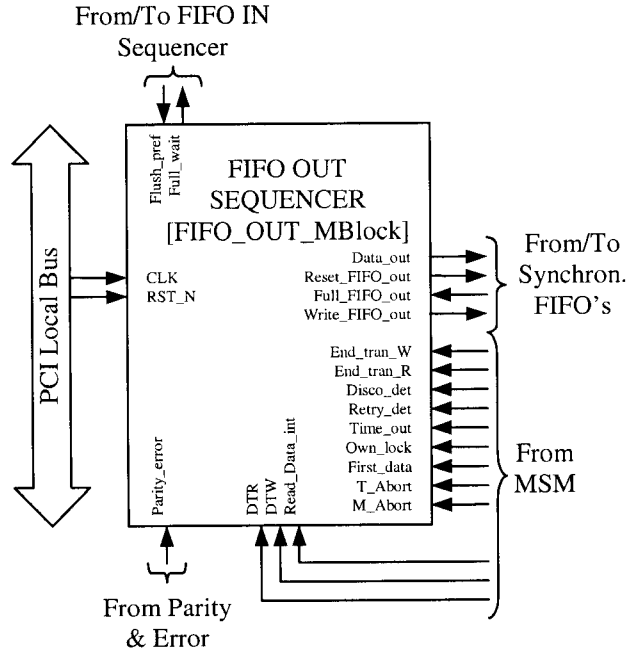


Figure 19. FIFO OUT Sequencer [FIFO_OUT_Mblock].

This block is basically a set of multiplexers that, depending on the information received from the MSM, select the bits to be sent to the Output Synchronisation FIFO [F_OUT_M].

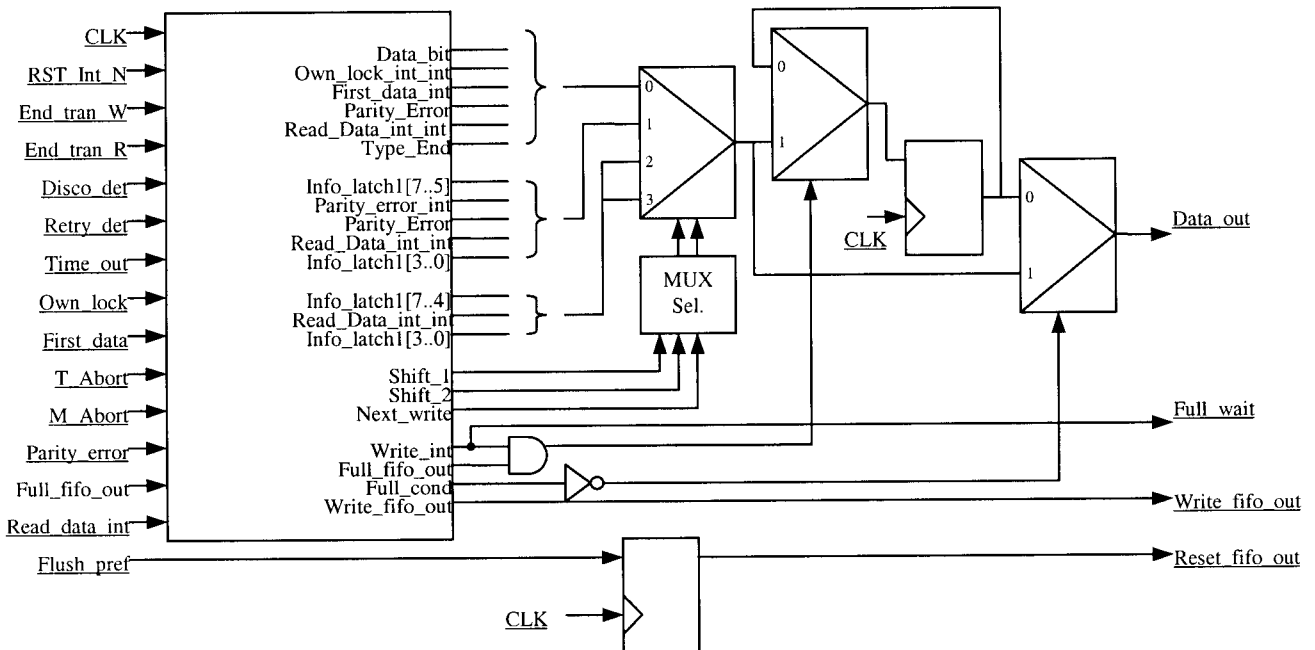


Figure 20. Schematics of the FIFO OUT Sequencer Block [FIFO_OUT_Mblock].

Master State Machine

Analogous to the Target State Machine (TSM) in the Target block of the PCI Core, this sub-block implements the main functions in the Master block of the PCI Core. Once it has received the data from the FIFO IN Sequencer or from the PCI Bus, it produces the appropriate signals to command the rest of the sub-blocks when the back-end application becomes the master of a bus transaction. Figure 21 shows the interconnections of the MSM block with the other blocks in the PCI Core Master.

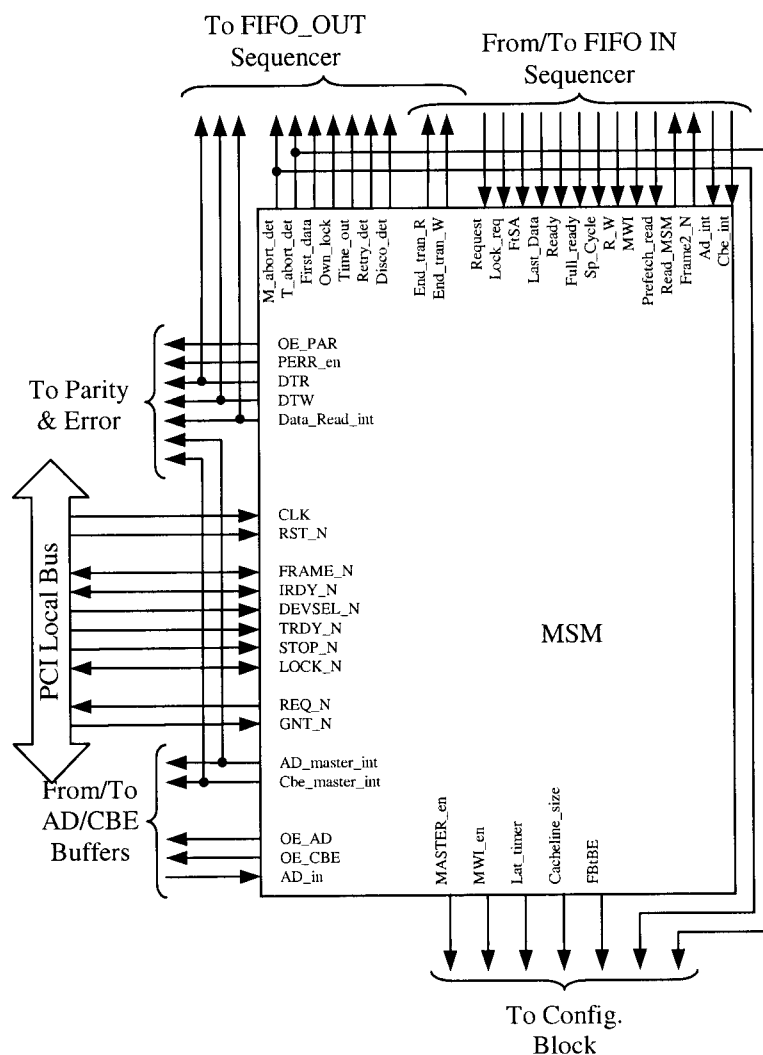


Figure 21. Master State Machine Block.

The main part of the architecture of the MSM block implements the two state diagrams shown in figures 22 and 23. A description of the states can be found in [1].

The main state diagram (figure 22) generates the signal lines that command the rest of the blocks following the PCI Specification.

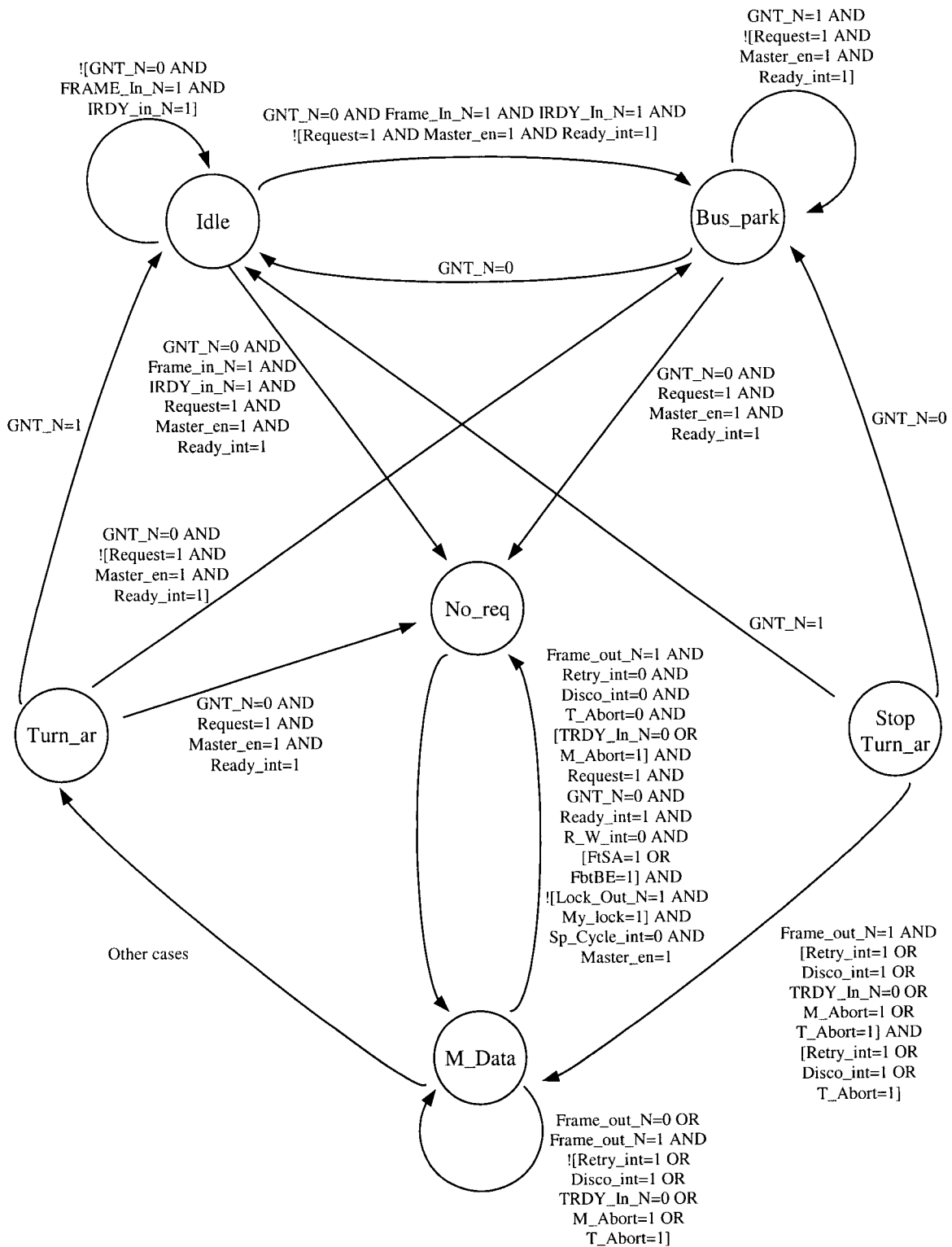


Figure 22. Main state diagram in the Master State Machine.

A Lock state machine (figure 23) is needed to avoid more than one master taking control of the LOCK# line.

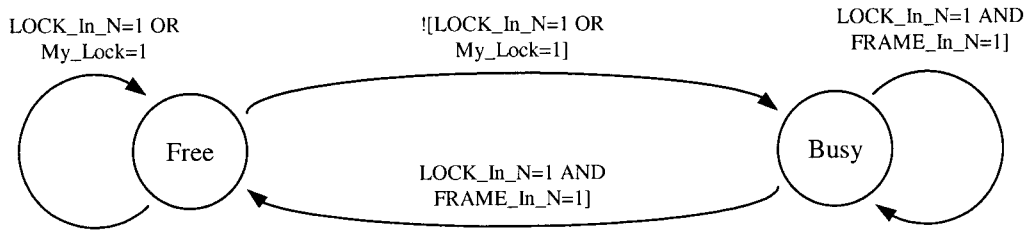


Figure 23. Lock State diagram.

AD/CBE Buffers Master

The AD/CBE buffers connect the 32-bit address/data and the 4-bit Command/Byte Enable lines of the internal buses to the PCI Local Bus.

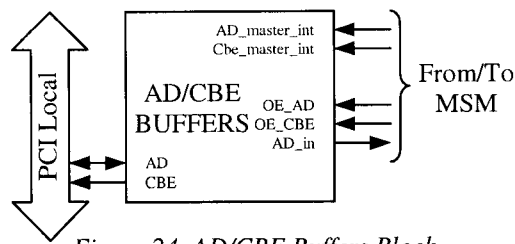


Figure 24. AD/CBE Buffers Block.

This block is controlled by the MSM signal lines which activate or deactivate the output buffers according to the cycle in process.

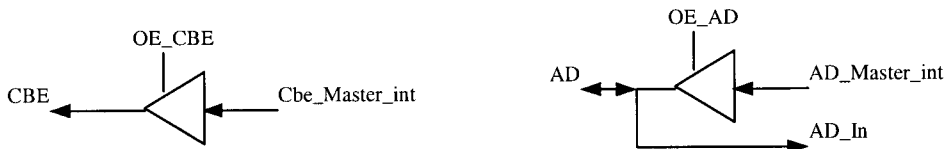


Figure 25. Schematics of the AD/CBE Buffers Master Block.

Parity&Error Master

This block computes parity and reports parity errors when the PCI Core is working as Master.

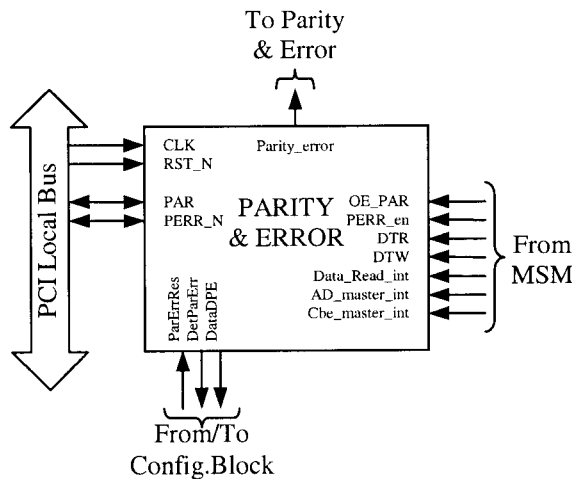


Figure 26. Parity & Error Block.

Figure 27 shows the schematics in the Parity and Error block. The parity signal is generated by this block for outgoing data. For incoming data, this block works as a parity checker.

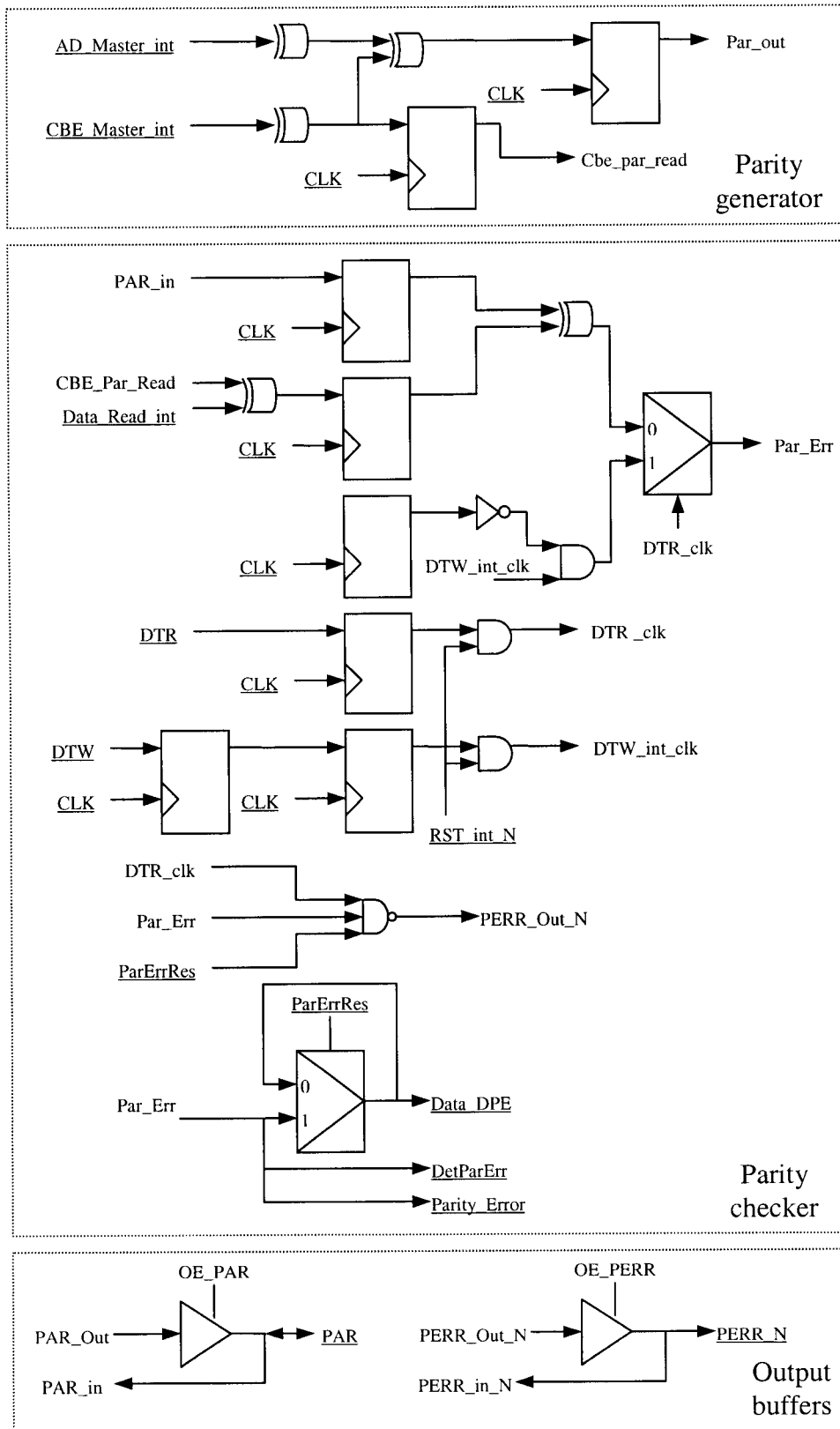


Figure 27. Schematics of the Parity & Error Master Block.

According to the PCI specification, the signals PAR and PERR_N should be provided respectively one and two clocks later than the data they refer to.

2.2.3.- Configuration block

The Configuration Block contains the configuration space registers and handles read/write access to them. It also manages the address decoding for Memory and I/O cycles.

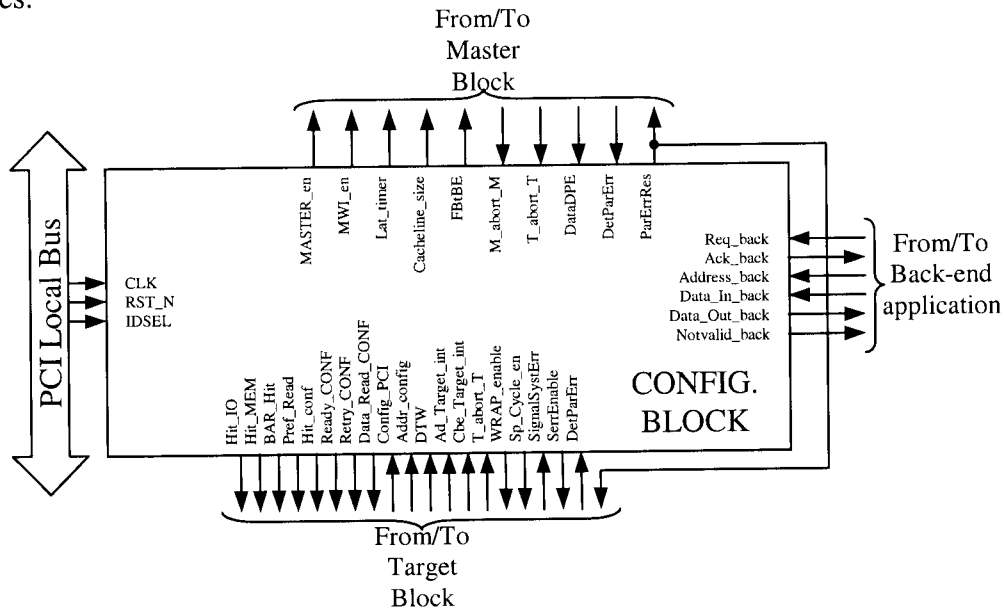


Figure 28. Configuration Block.

An extra function has been added to this block: the configuration cycle can be performed by the back-end application using the Local Configuration Lines (ending in '_back' in figure 28). This is not included in the PCI Specification but is compatible with it, so back-end applications provided with this mechanism can profit from it.

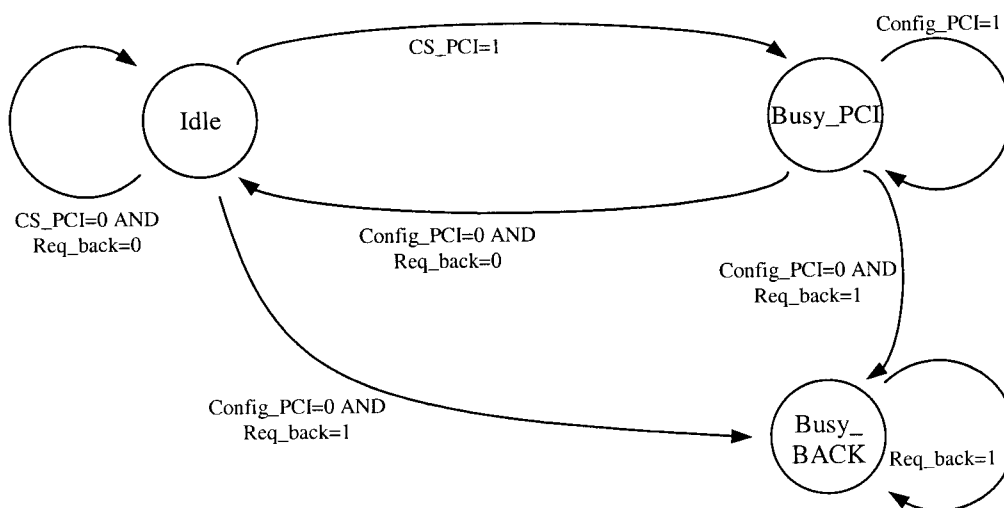


Figure 29. Configuration Block State Diagram.

The physical addresses of the Configuration Space and its contents is explained in [1].

3.- Simulations

A description of several simulations of the PCI Interface now follows. Given the amount of possible combinations, only a few relevant simulations are shown to allow an understanding of its functionality.

The simulation system implements a PCI environment connecting two back-end applications to a PCI Local Bus through its corresponding PCI Interfaces (see figure 30)

As explained in section 2.1, the FIFO's designed by Riccardo Locatelli have been substituted for behavioural models in VHDL.

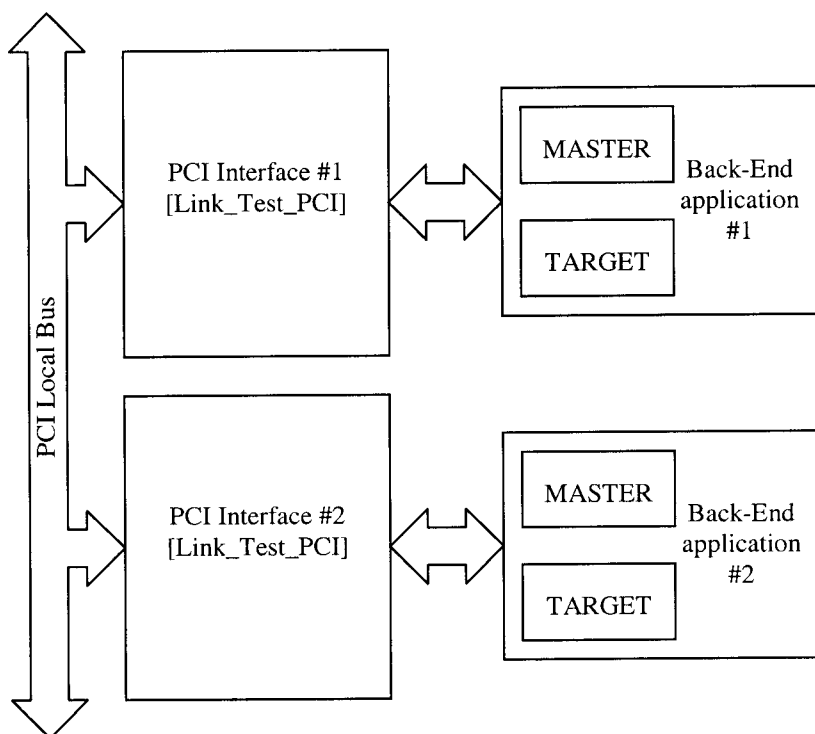


Figure 30. PCI Interface connection for testing.

3.1.- Write cycles

There are several possible write cycles: memory write, I/O write, memory write multiple, memory write line. Here only the single memory write is described thoroughly as it is the base for the other cycles.

3.1.1.- Single Memory Write Cycle

To write one single data in back-end application #2, back-end application #1 sends the following information to the PCI Interface: In the first word, the master sends the encoded command (0111 = *Memory Write*), the address where the data has to be written and the transaction type (1111 = *Address Single*). In the second word the master sends the Byte Enable Lines (inverted logic), the data to be written and the transaction type (1000 = *Last Data*).

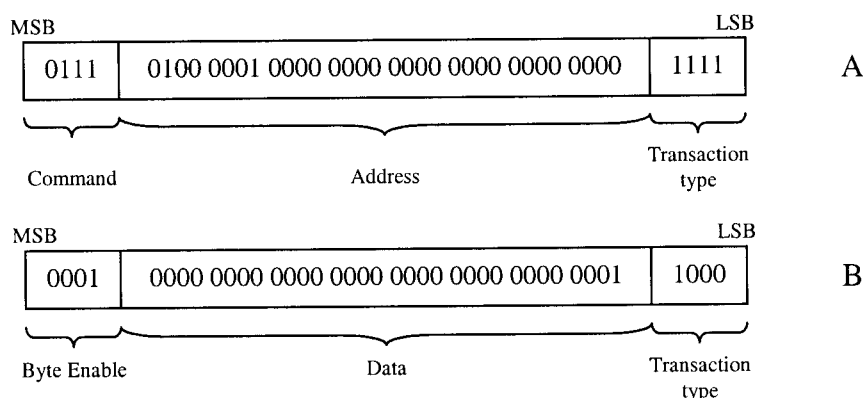


Figure 31. Words sent by the Back-end Application #1.

The Single Memory Write cycle starts with the writing of the two words (A and B) by the back-end application #1 into the PCI Interface (Data_in_IN_M). The PCI Interface decodes the Command and Transaction Type of the first word and requests the bus by asserting the REQ_N signal, indicating there is a transaction to be made. When it has the bus grant (GNT_N = 0), the master starts the address phase by placing a valid address(C) and command (F) on AD and C/BE# signal lines respectively and asserting the FRAME_N signal (inverted logic). The FRAME_N signal is deasserted at the end of the Address phase.

The Data phase then starts by placing the Data (D) and the Byte Enable Lines (F) on AD and C/BE# signal lines respectively and asserting IRDY_N. As the Parity signal is delayed by one clock the odd parity for the Address phase is now set in the PAR line. The back-end application decodes the address and claims the access by asserting the DEVSEL_N signal line. The access of data occurs on the rising edge of the clock signal line when both the IRDY_N and TRDY_N signal lines are asserted.

The PCI Interface #2 indicates to back-end application #2 that there is data in the Output Synchronising FIFO (F_OUT_T) by asserting the rdempty_OUT_T line. The back-end application requests the data (rdreq_OUT_T=1) and the PCI Interface sends the data.

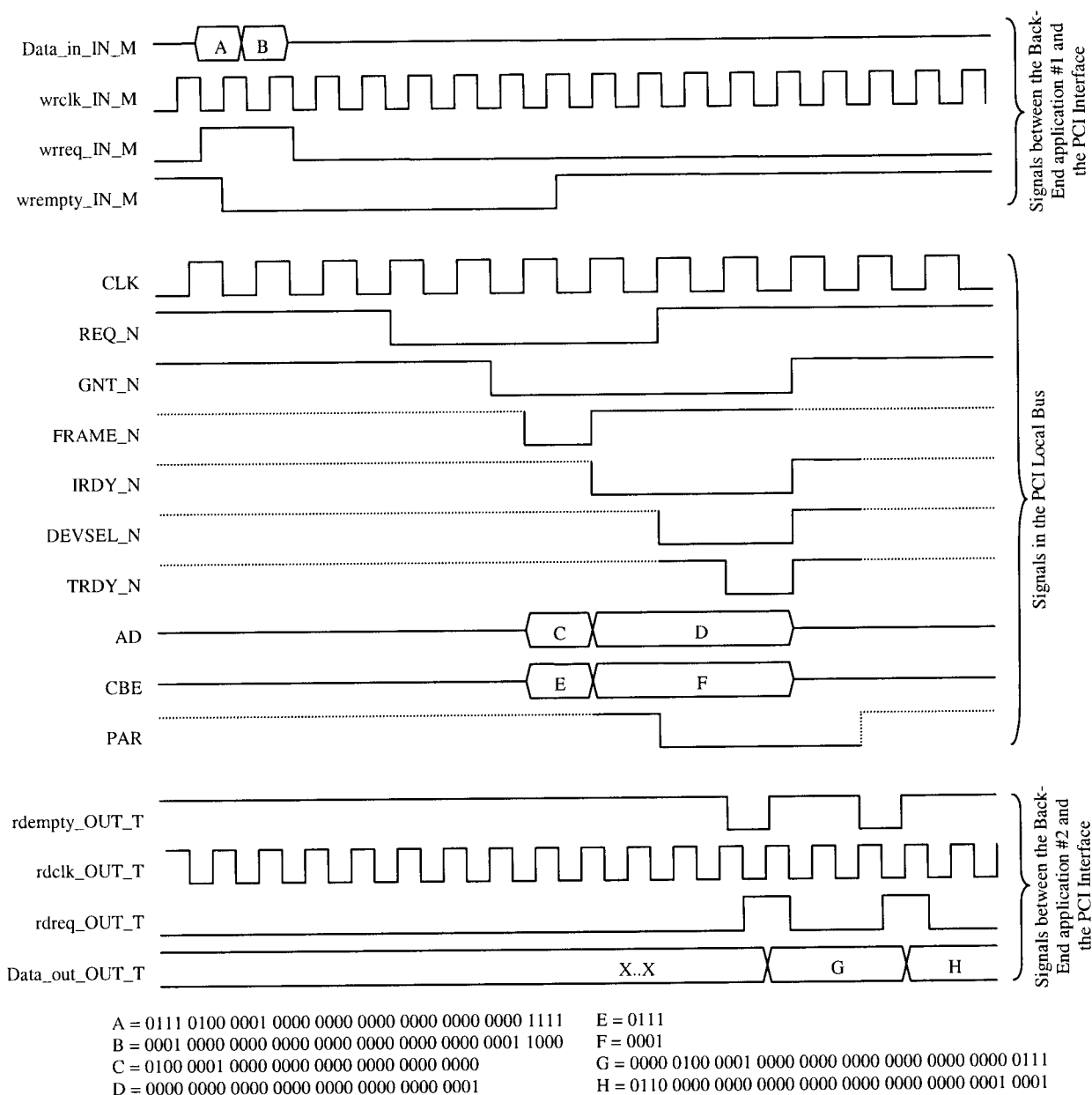


Figure 32. Timing diagram of a Single Write Cycle.

Note that the order in which the back-end application receives the data (G) is not the same in which it was sent (A): the command that was encoded in the four most significant bits by back-end application #1 (master), is received by back-end application #2 (target) in the four least significant bits.

3.1.2.- Burst Write Cycle

A burst write cycle follows the same pattern as a single write cycle with the only difference that after the address phase (A), more than one data word follows. The least significant bits in the data are set to 0100 (0100 = Data) except in the last data (D) where it is set to 1000 (Last Data) as shown in figure 33.

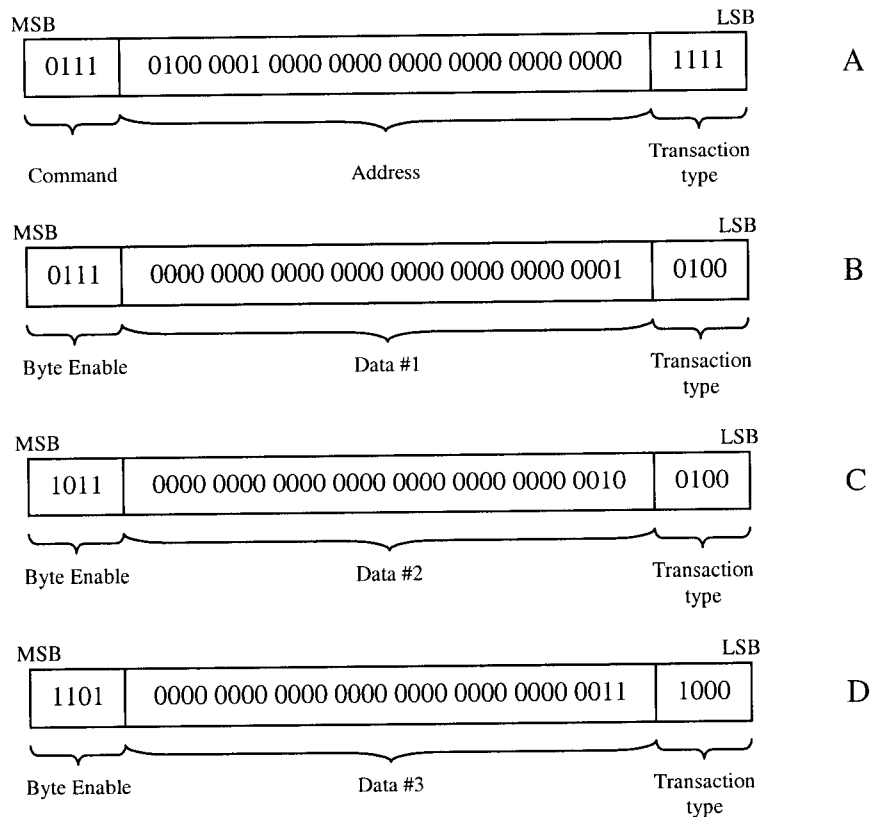


Figure 33. Words sent by the Back-end Application #1.

3.1.3.- I/O Write Cycle

An I/O Write cycle works in the same way as a memory write cycle, but the address corresponds to an application mapped in I/O Address Space.

3.2.- Read cycles

As with the write cycles, not every possible combination of read access is analysed in this section.

3.2.1.- Single Memory Read Cycle

To read one single data word from back-end application #2, back-end application #1 sends the following information to the PCI Interface: In the first word, the master sends the encoded command (0110 = *Memory Read*), the address to be read and the transaction type (1111 = *Address Single*). In the second word the master sends the Byte Enable lines (inverted logic) and the transaction type (1000 = *Last Data*).

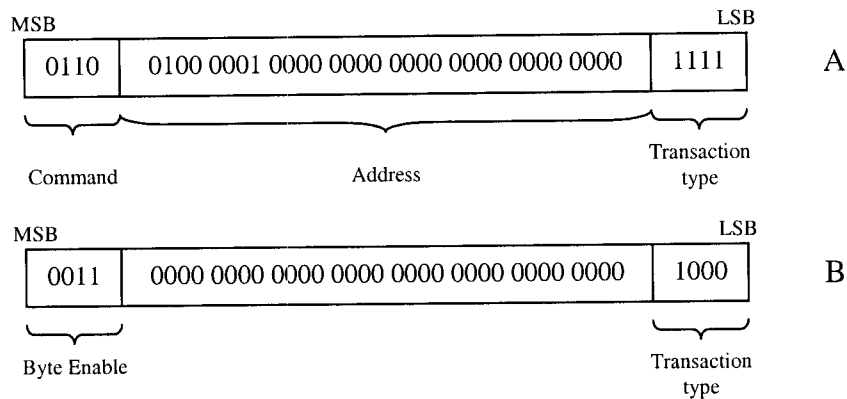


Figure 34. Words sent by back-end application #1.

The Single Memory Read cycle starts with the writing of the two words (A and B) by the back-end application #1 into the PCI Interface (Data_in_IN_M). The PCI Interface decodes the Command and Transaction Type of the first word and requests the bus by asserting the REQ_N signal, indicating there is a transaction to be made. When it has the bus grant (GNT_N = 0), the master starts the address phase by placing a valid address (C) and command (F) on AD and C/BE# signal lines respectively and asserting the FRAME_N signal (inverted logic). The FRAME_N signal is deasserted at the end of the Address phase.

The PCI Bus Master (Back-end application #1) then drives valid byte enable information onto the C/BE# signal lines and the parity of the address (C) and command (F) onto the PAR signal line. The information is transferred in that way to PCI Interface #2.

At the target side of the transaction, the behaviour differs depending on the value of the Memory Prefetchable Status bit in the BAR's of the Configuration Space: when this bit is set to 0, meaning the memory space is not prefetchable, the PCI Interface #2 passes two words to the back-end application #2 (figure 35); on the contrary if the Memory Prefetchable Status bit is set to 1, indicating the memory space is prefetchable, the PCI Interface #2 passes only one word to the back-end application (figure 36), to which the back-end application should respond without waiting for the BE lines.

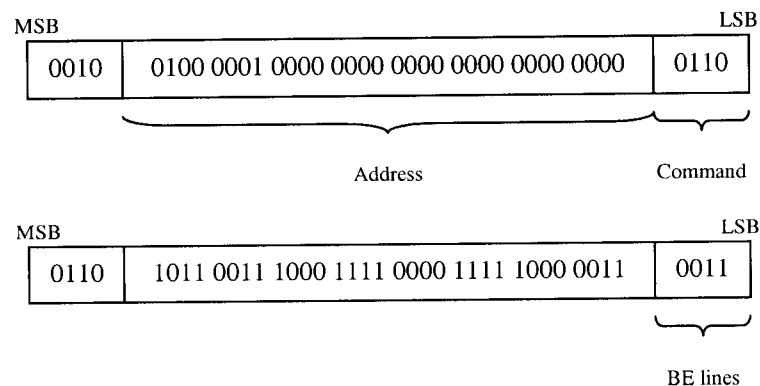


Figure 35. 40-bit words received by a target (back-end application #2) with Memory Prefetchable Status bit set to 0.

The timing shown in figure 40 corresponds to this second case where the memory space is prefetchable: the signal `rdempty_OUT_T` in PCI Interface #2 is set to zero indicating that the information has arrived to the Output Synchronisation FIFO (`F_OUT_T`). The target gets this information containing the address to be read via the `Data_out_OUT_T` line (*H*). Once the address is read, the Output Synchronisation FIFO sets the empty flag to one indicating there is no more data to be read. The target answers by placing the data in `Data_in_IN_T` (*J*). The data is placed in the lowest 32 bits, and the next two bits are used as Stop and `T_Abort` lines (see figure 37).

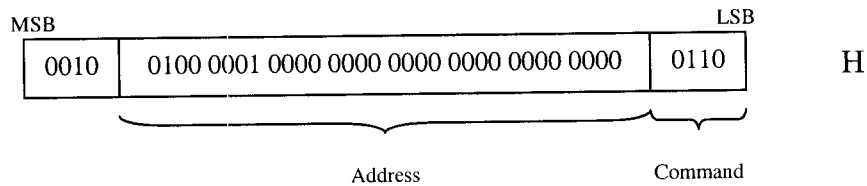


Figure 36. 40-bit word received by a target (back-end application #2) with Memory Prefetchable Status bit set to 1.

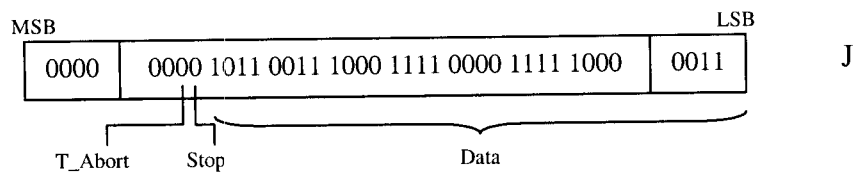


Figure 37. 40-bit word sent by back-end application #2.

Once the data is ready for sending, PCI Interface #2 asserts `TRDY_N`. The access of data by PCI Interface #1 occurs on the rising edge of the `CLK` signal line when both the `IRDY_N` and `TRDY_N` signal lines are asserted. This data reaches back-end application #1 via `Data_out_OUT_M` (*K*) together with the 'End-type information'.

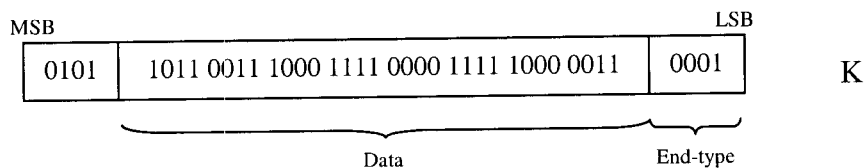


Figure 38. 40-bit word received by back-end application #1.

A last data (*I*) is sent to the target via `Data_out_OUT_T` to indicate the conclusion of the transaction

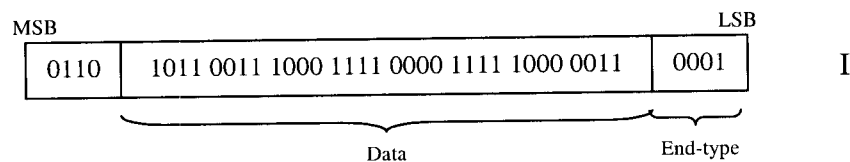


Figure 39. 40-bit word received by back-end application #2.

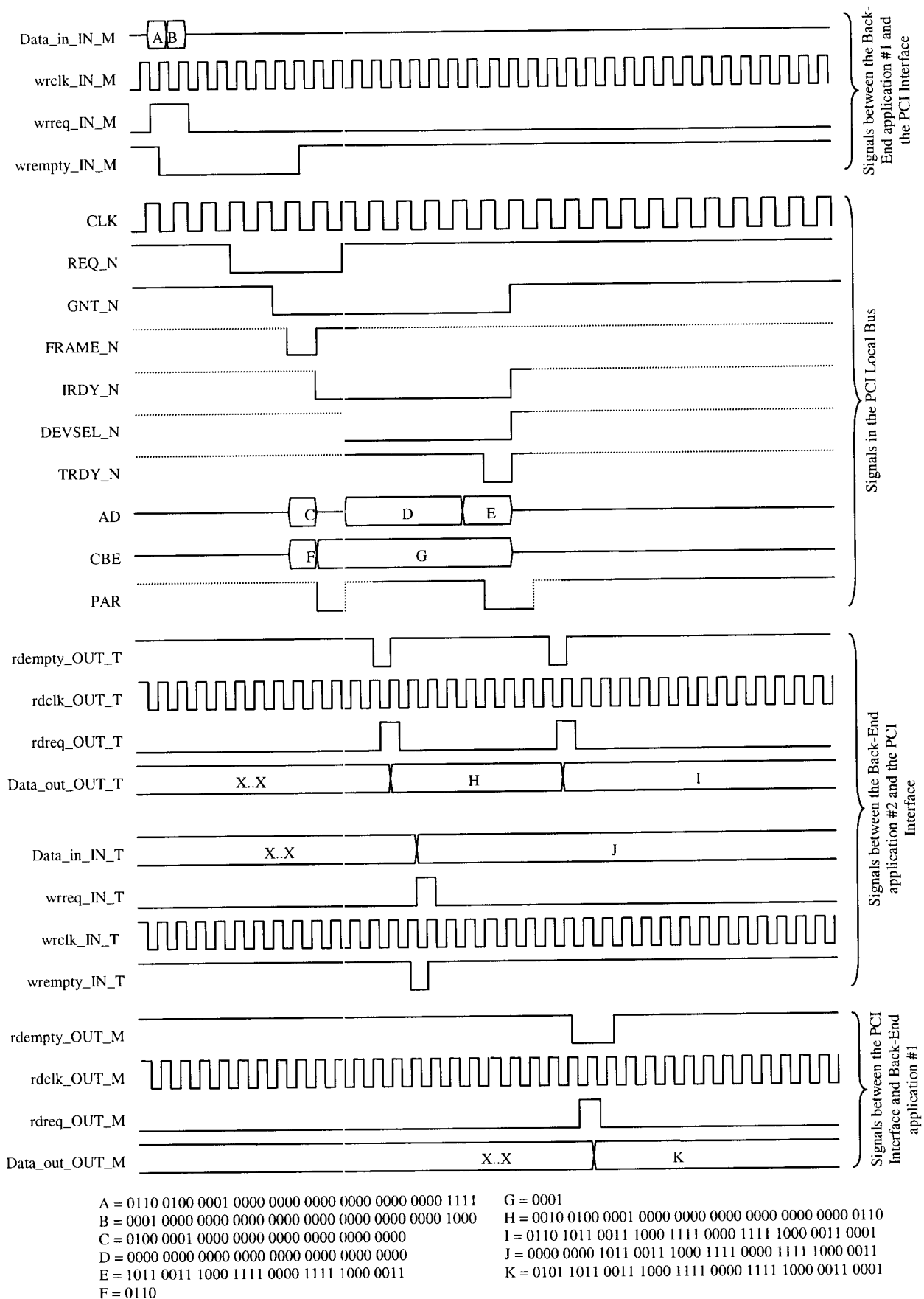


Figure 40. Timing diagram of a Single Read Cycle to a pretetchable memory space.

3.2.2.- Burst Read Cycle

The Burst Read Cycle starts as a Single Read Cycle with the exception that after the address, the Byte Enable lines have to be sent (see figure 41).

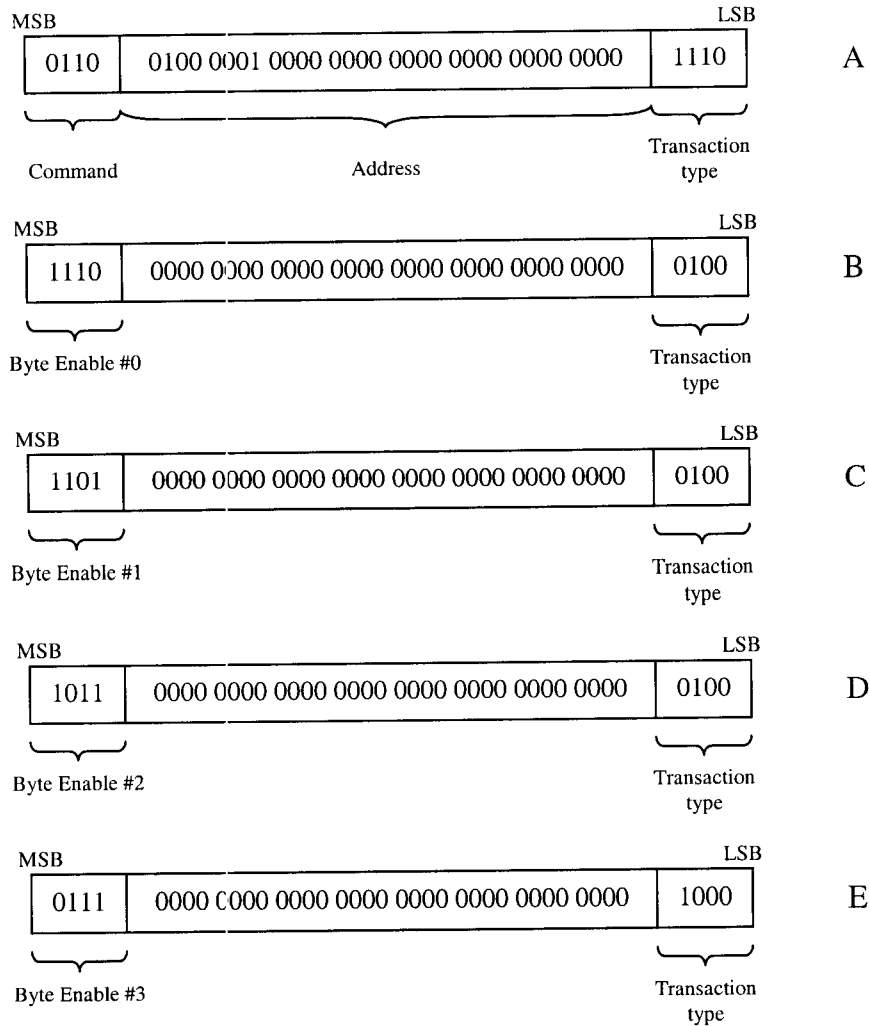


Figure 41. Words sent by back-end application #1.

In this example (shown in figure 42), word A contains the command (0110 = *Memory Read*), the address to be read and the transactions type (1110 = *Address Burst*). Words B through E contain the Byte Enable Lines (in inverted logic) and the transaction type (0110 = *Data*, 1000 = *Last Data*)

As in the Single Read Cycle, the target only receives the first word (*M*) and the target has been forced to place four data words as response to the request (*O*, *P*, *Q* and *R*).

The master only receives one of these data (*S*, *T*) because the target has stopped the transaction (*STOP_N* = 0). The cause of this stop signal is the timer having reached its maximum which is set in 8 cycles.

As in the Single Read Cycle, the target receives the data that it has sent itself earlier (*N*).

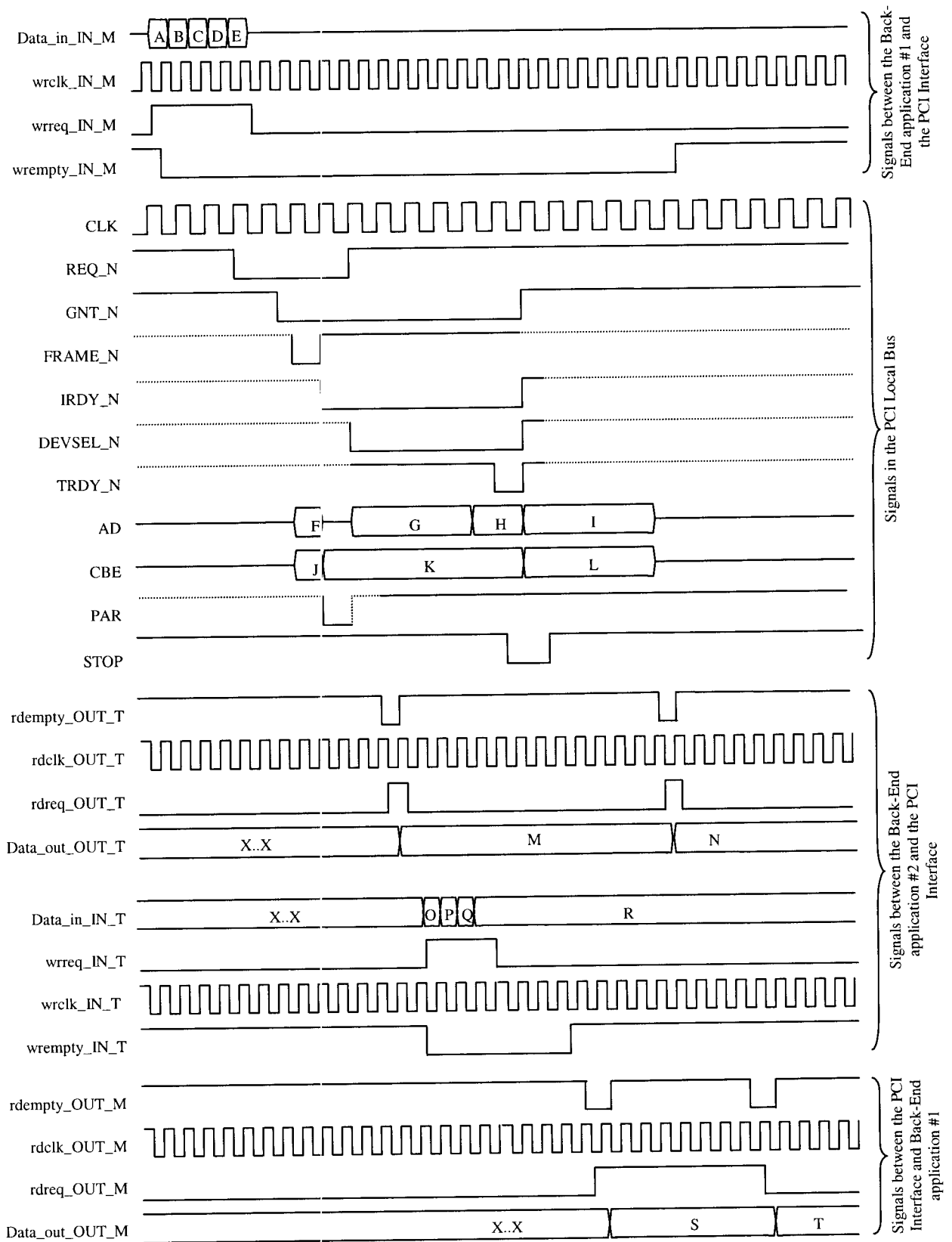


Figure 42. Timing diagram of a Burst Read Cycle.

A = 0110 0100 0001 0000 0000 0000 0000 0000 0000 1110	K = 1110
B = 1110 0000 0000 0000 0000 0000 0000 0000 0000 0100	L = 1101
C = 1101 0000 0000 0000 0000 0000 0000 0000 0000 0100	M = 0011 0100 0010 0000 0000 0000 0000 0000 0000 0110
D = 1011 0000 0000 0000 0000 0000 0000 0000 0000 0100	N = 0111 1111 1111 0000 0000 1111 1111 0000 0000 1101
E = 0111 0000 0000 0000 0000 0000 0000 0000 0000 1000	O = 0000 0000 1111 1111 0000 0000 1111 1111 0000 0000
F = 0100 0001 0000 0000 0000 0000 0000 0000 0000	P = 0000 0000 1010 1010 1010 1010 1010 1010 1010 1010
G = 0000 0000 0000 0000 0000 0000 0000 0000 0000	Q = 0000 0000 1100 1100 1100 1100 1100 1100 1100 1100
H = 1111 1111 0000 0000 1111 1111 0000 0000	R = 0000 0000 1000 1000 1000 1000 1000 1000 1000 1000
I = 1010 1010 1010 1010 1010 1010 1010 1010 1010	S = 1010 1111 1111 0000 0000 1111 1111 0000 0000 0000
J = 0110	T = 0000 1111 1111 0000 0000 1111 1111 0000 0000 0101

Figure 43. Data Exchange in the Burst Read Cycle.

3.2.3.- I/O Read Cycle

An I/O read cycle works in same way as a memory read cycle, but the address corresponds to an application mapped in I/O Address Space.

3.3.- Configuration cycle

The purpose of the configuration cycle shown in this example (figure 45) is to write the BAR registers of the back-end application #2. The agent that performs the configuration is back-end application #1 by sending the following 40-bit words:

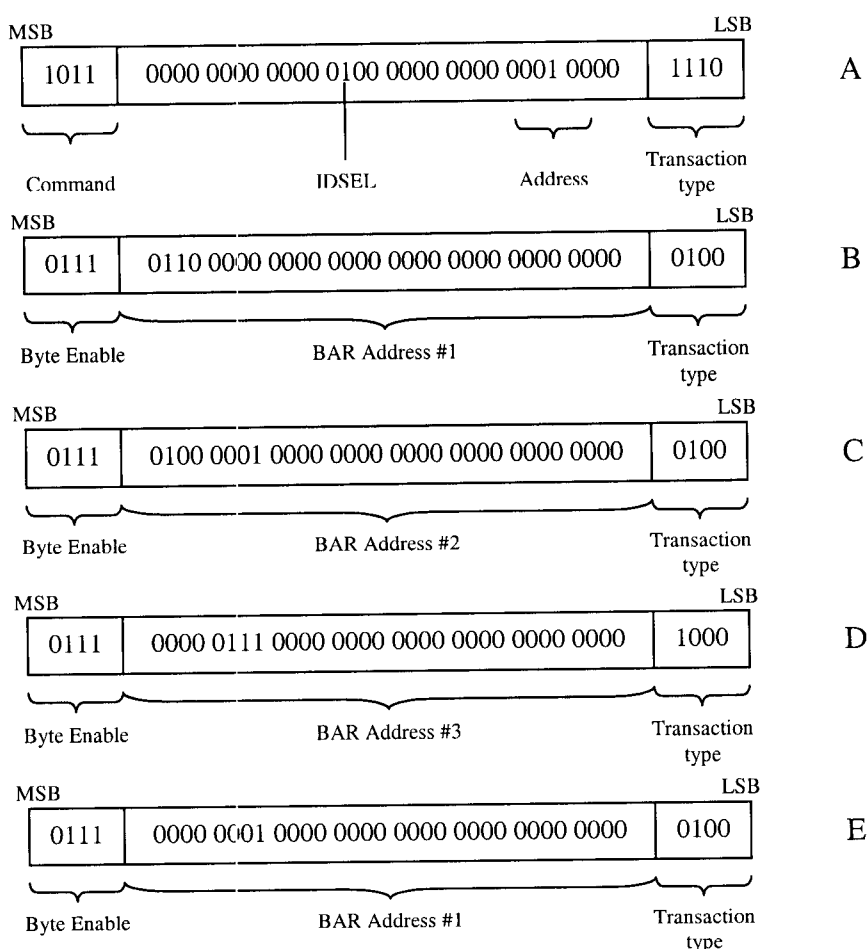


Figure 44. Words sent by back-end application #1.

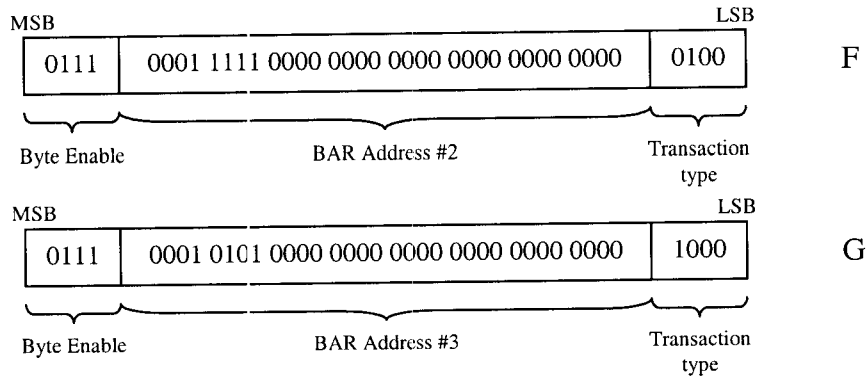


Figure 44. Words sent by back-end application #1 (continuation).

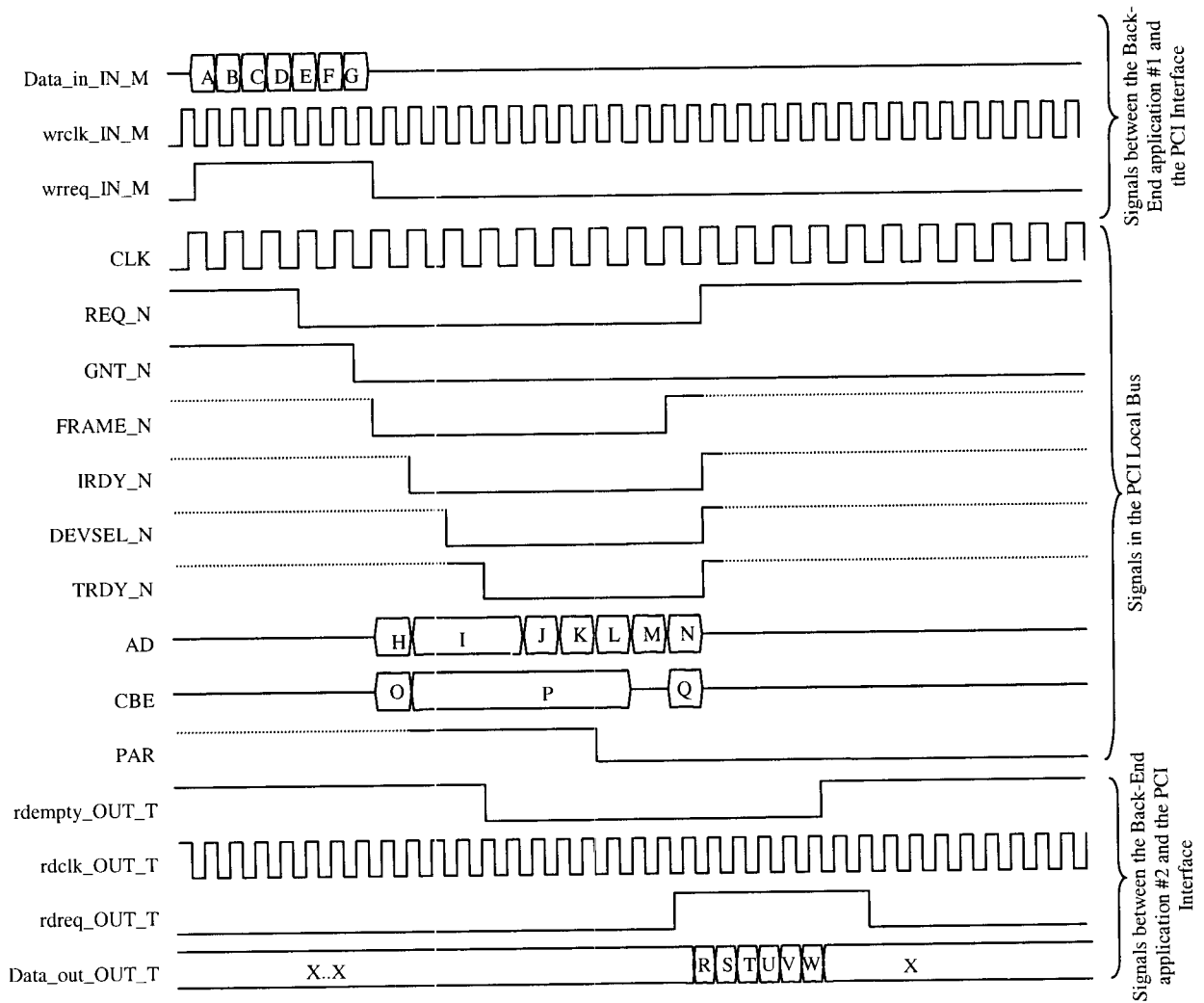


Figure 45. Timing diagram of a Configuration Write Cycle.

```

A = 1011 0000 0000 0000 0100 0000 0000 0001 0000 1110
B = 0111 0110 0000 0000 0000 0000 0000 0000 0000 0100
C = 0111 0100 0001 0000 0000 0000 0000 0000 0000 0100
D = 0111 0000 0111 0000 0000 0000 0000 0000 0000 0100
E = 0111 0000 0001 0000 0000 0000 0000 0000 0000 0100
F = 0111 0001 1111 0000 0000 0000 0000 0000 0000 0100
G = 0111 0001 0101 0000 0000 0000 0000 0000 0000 1000
H = 0000 0000 0000 0100 0000 0000 0001 0000
I = 0110 0000 0000 0000 0000 0000 0000 0000
J = 0100 0001 0000 0000 0000 0000 0000 0000
K = 0000 0111 0000 0000 0000 0000 0000 0000
L = 0000 0001 0000 0000 0000 0000 0000 0000
M = 0001 1111 0000 0000 0000 0000 0000 0000
N = 0001 0101 0000 0000 0000 0000 0000 0000
O = 1011
P = 0111
Q = 0111
R = 0001 0000 0000 0000 0100 0000 0000 0001 0000 1011
S = 0101 0110 0000 0000 0000 0000 0000 0000 0000 0111
T = 0101 0100 0001 0000 0000 0000 0000 0000 0000 0111
U = 0101 0000 0111 0000 0000 0000 0000 0000 0000 0111
V = 0101 0000 0001 0000 0000 0000 0000 0000 0000 0111
W = 0101 0001 1111 0000 0000 0000 0000 0000 0000 0111
X = 0110 0001 0101 0000 0000 0000 0000 0000 0000 0111

```

Figure 46. Data Exchange in the Configuration Write Cycle.

In the first word (A) the master sends the encoded command (1011 = Configuration Write), the selected agent (AD[18] is connected to the IDSEL signal line of the PCI Interface #2), the address where the data has to be written (000100 is the address of the first Base Address Register) and the transaction type (11101 = Address Burst). The following words contain the data to be written in the BAR's.

The data is written in the Base Address Registers with the characteristic that only the most significant byte is recorded. This is not because only the first Byte Enable line has been selected in all the words, but because it has been programmed that way in the VHDL files. As a result, the first byte is that written in the configuration cycle, and the three last bytes are those fixed by hardware (see figure 47).

0110 0000	0000 0000	0000 0000	0000 1000	10 _h
0100 0001	0000 0000	0000 0000	0000 1000	14 _h
0000 0111	0000 0000	0000 0000	0000 0000	18 _h
0000 0001	0000 0000	0000 0000	0000 1000	1C _h
0001 1111	0000 0000	0000 0000	0000 0001	20 _h
0001 0101	0000 0000	0000 0000	0000 0001	24 _h

⏟
Programmable
⏟
Fixed by hardware

Figure 47. BAR's in the Configuration Space after the Write Configuration Cycle shown earlier.

The Local Configuration Cycle (from back-end application) is described in detail in [1].

4.- Conclusions

Simulation of the design has been done and it has proved to work correctly in most cases. There are nevertheless a few points that need comment.

1.- Bits structure.

The position of the bits is not the same during the cycles. As it has been seen in the single memory write and single memory read cycles, the master sends the command in the four most significant bits and the target receives it in the four least significant bits. The same happens with the position of the data read in the read cycle: the target must place the data in the 32 lowest bits while the master (and target) receives it in the 32 middle bits.

Although the system works correctly, it would be clearer for operation and understanding if the bits would keep the same position throughout the entire cycle.

2.- Read Cycle.

As explained in more detail in section 3.2, the read cycle does not complete as expected. Only the address to be read is received by the target. It takes sixteen clock cycles more to receive the Byte Enable lines, resulting in the target asserting the Stop signal line due to latency timer expiration. To study the rest of the read cycle, the target has been forced to answer by writing data in the corresponding Synchronisation FIFO as soon as the first word is received. This data reaches the master correctly.

In the case of a burst read cycle, the transaction is not finished, even forcing data into the Synchronisation FIFO as soon as the address is received, due to the expiration of the target latency timer.

3.- Target receives its own data.

When a read cycle is performed, the requested data is sent to the master but also returned to the same target that sent it. This problem has also been observed in the Configuration cycle: the data sent to the Configuration Space of the PCI Interface is also received by the target of the back-end application.

4.- Long cycles.

According to the PCI Specification [3], a standard write cycle lasts two or three clock periods (see figure 48). A typical write cycle in the system under study lasts a minimum of four clock periods (see figure 32).

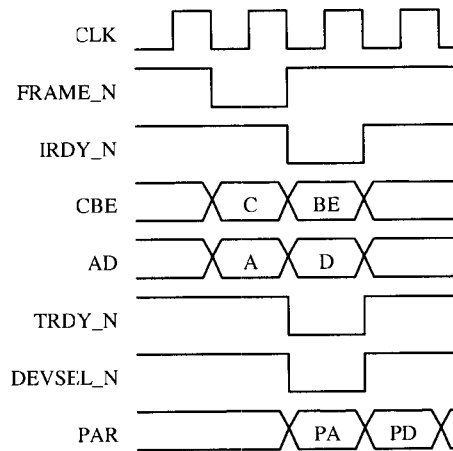


Figure 48. Standard Write Single Access Cycle.

Similarly, a standard read cycle lasts three or four clock periods according to PCI Specifications (see figure 49) while in the system under study it takes a minimum of five clock periods (see figure 40) without waiting for the Byte Enable lines.

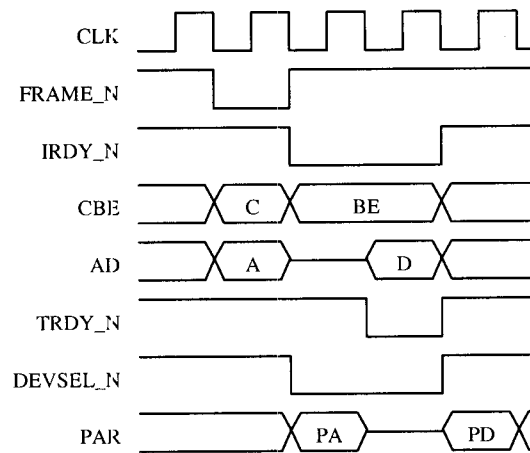


Figure 49. Standard Read Single Access Cycle.

5.- Not fully configurable BAR's.

On a Configuration Write Cycle, only the eight most significant bits of the Base Address Registers (BAR's) can be modified. The other 24 bits are fixed by hardware.

References

- [1] R. Locatelli, *Master/Target PCI VHDL Core*, ESA 1999.
- [2] *PCI Local Bus Specification*, PCI Special Interest Group 1995.
- [3] E. Solari, G. Willse, *PCI Hardware and Software. Architecture & Design*, Annabooks 1996.