INTERNATIONAL STANDARD

ISO/IEC 14576

First edition
1999-12

**Information technology –
Synchronous split transfer type system bus (STbus) – Logical layer**

*Technologies de l'information –
Bus de système de transfert de fente synchrone (STbus) –
Couche logique*

Reference number
ISO/IEC 14576:1999(E)

INTERNATIONAL STANDARD

ISO/IEC 14576

First edition
1999-12

Information technology – Synchronous split transfer type system bus (STbus) – Logical layer

Technologies de l'information –
Bus de système de transfert de fente synchrone (STbus) –
Couche logique

PRICE CODE X

For price, see current catalogue

Contents

Figures

Tables

INFORMATION TECHNOLOGY –

SYNCHRONOUS SPLIT TRANSFER TYPE SYSTEM BUS (STbus) –

LOGICAL LAYER

FOREWORD

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 14576 was prepared by subcommittee 26: Microprocessor systems, of ISO/IEC joint technical committee 1: Information technology.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

Annexes A, B and C are for information only.

INFORMATION TECHNOLOGY – SYNCHRONOUS SPLIT TRANSFER TYPE SYSTEM BUS (STbus) – LOGICAL LAYER

1. Overview

1.1 Scope

This International Standard specifies the logical specifications of STbus which is a high-performance and highly reliable system bus. STbus adopts a synchronous transfer method with a high-speed clock and a split transfer method enabling to minimize bus holding time during one bus operation and to use a bus efficiently.

The contents given in this specifications are as follows:

a) System bus interface signal provisions;

b) Bus operations and transfer protocol for each bus operation;

c) Copyback cache coherency control for maintaining consistency between a shared memory and a cache memory of each processor in a multiprocessor system;

d) Fault detection function using parity check and duplex configuration for control signals.

1.2 Applicability

This International Standard is Applicable to a high-performance system bus or an I/O bus in a multiprocessor system. Typical STbus applications are indicated in Figure 1:

a) A System bus and an I/O bus in a TCMP system;

b) A System bus in an LCMP system.

- TCMP: tightly coupled multiprocessor system
  (A system consisting of two or more processors sharing the same memory, with the entire system controlled by one OS.)
- LCMP: loosely coupled multiprocessor system
  (A system in which each processor is connected by a shared memory or other medium, with each processor operated by an individual OS.)

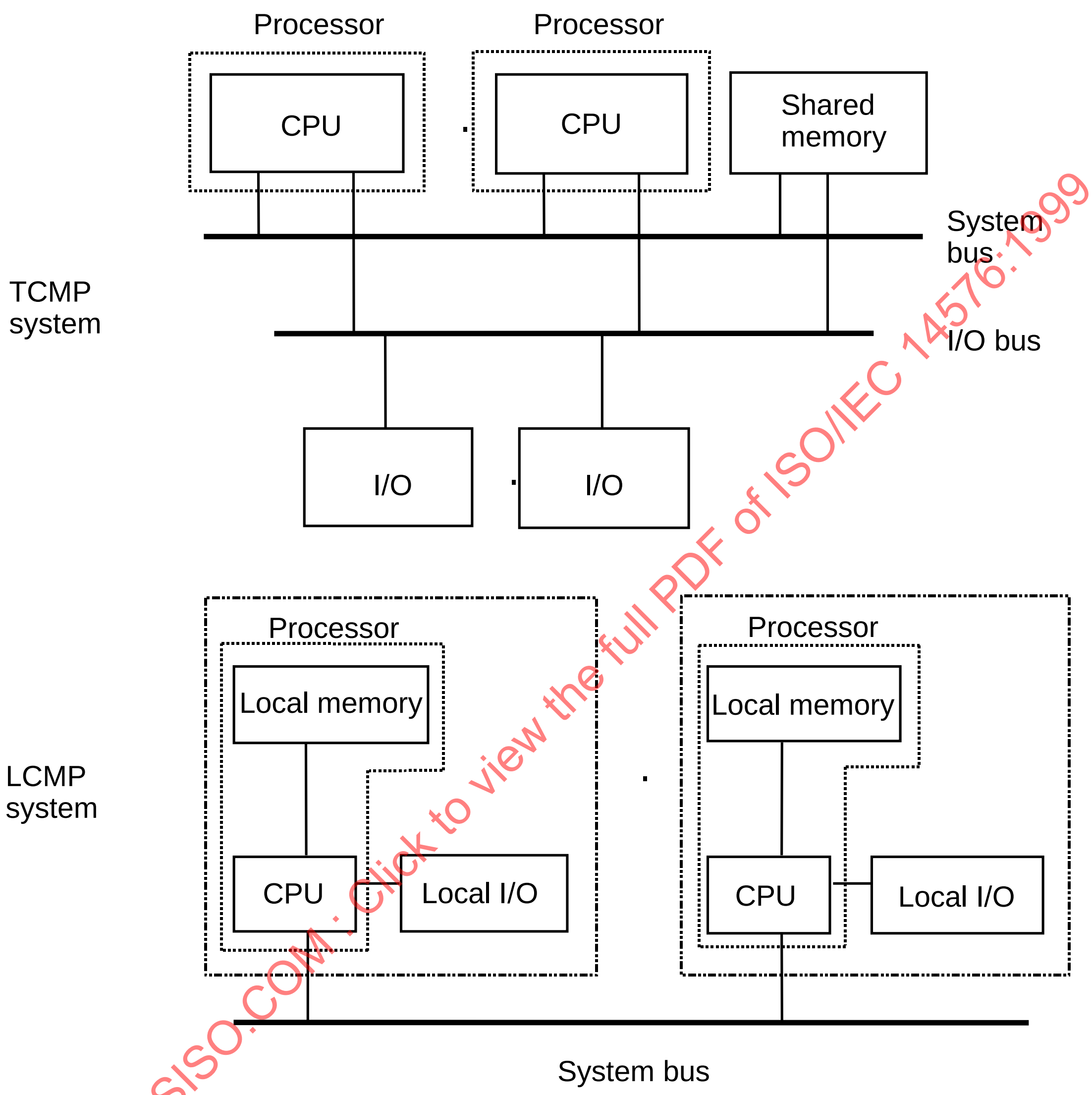


Figure 1 - STbus Applications

2. Definitions

2.1 Explanation of Terms

For the purposes of this International Standard, the following terms and definitions apply.

1) Answer transaction

An information transfer operation by which a function unit receiving a command returns answer information, to notify the unit issuing the command that the command has been completed (in some cases the requested data is appended) and to indicate status information.

2) Basic signal

Those bus interface signals that must be implemented in every STbus system, and thus for which compatibility is assured among different systems.

3) Block

The minimum unit registered in cache memory. In STbus this is limited to 32 bytes.

4) Bus handler (BH)

A concentrated bus control mechanism for sorting out competing bus requests from different function units, selecting one of the requests, and granting the bus right to that function unit.

5) Bus master

A function unit that has the bus right (a grant signal has been asserted) and is transferring information on the bus.

6) Bus slave

A function unit to which information is being transferred by the bus master.

7) Bus snoop

Monitoring of the bus for read operations from external memory and write operations to external memory.

8) Cache invalidation

A request to invalidate a block in cache memory. For example, when a write access is made to a Shared & Unmodified (SU) area, this is used to invalidate the same area in another cache.

9) CPU

A central processing element with functions for interpreting and executing instructions. In these specifications, cache memory is included with the CPU.

10) Copyback scheme

A cache updating method in which data written by the processor or instruction execution part is updated only in the cache, without being reflected directly in memory. The copyback cache supported in STbus has the following three internal states: Invalid state (I), Shared & Unmodified state (SU), Exclusive & Modified state (EM).

11) DUT (Destination Unit)

A function unit performing an answer transaction.

12) Exclusive & Modified state (EM)

An internal state in a copyback cache, whereby the only place in the system an access area is registered is in cache memory, and the contents are not the same as shared memory. In this state, only the cache has been updated.

13) Function unit

A hardware unit connected to the bus and having a mechanism for bus interface control. Normally one function unit consists of one board.

14) I/O adapter

A function unit that controls I/O devices under control of a processor.

15) Invalid state (I)

A state in which an area accessed by the processor is not registered in cache memory.

16) Modified read command

A command issued to the system bus by a copyback cache memory when a write access by the processor results in a write miss.

17) Optional signal

Those bus interface signals that users are free to adopt or not in system implementation.

18) Order transaction

An information transfer operation for sending a command and requesting processing by another function unit.

19) Parity

When not otherwise noted in these specifications, parity is always odd. Here odd parity means that when a given signal (e.g., 8 bits) is augmented by a parity bit (e.g., 8 + 1 = 9 bits), then if the sum of 1-bits in the augmented set is an even number (including 0) an error is detected.

20) Processor

A function unit with the capability of executing instructions and controlling the various I/O adapters. Processor consists of CPU and memory in general.

21) Read hit/read miss

When an instruction or operand to be read by the processor is registered in cache memory, this is called a read hit. If not, it is a read miss.

In the case of a read miss, if the object of the read is cacheable, one block containing the object is newly registered in the cache.

22) Retry indication

The temporary suspension of access by external devices to a copyback cache area that has been updated without the change having been reflected in main memory.

23) Shared & Unmodified state (SU)

An internal state in a write-through or copyback cache, whereby an access area is registered in a cache and has the same contents as shared memory. Sharing by more than one cache is possible.

24) SUT (Source Unit)

A function unit performing an order transaction.

25) Write hit/write miss

If an area to be written by the processor is registered in cache memory, this is called a write hit. If not, it is a write miss.

In the case of a write-through cache, the write data is immediately reflected in shared memory.

If a copyback cache scheme is used, in the case of a write hit the write data is reflected in the cache only. If a write miss occurs, one block of the write area is read from shared memory and newly registered, then the write data is written over that area in the cache only.

26) Write-through scheme

A cache updating method in which data written by the processor or instruction execution part is reflected directly in memory. The internal states are: Invalid state (I), Shared & Unmodified state (SU).

2.2 Notation

The following symbols and other notation are used in these specifications.

- Function unit numbers are indicated by (#n), and control signals to each unit are written as [signal line name + (function unit number)], e.g., RQL*(#n), GR*(#n).
- When the values of control signals are indicated, the following notation is used.
  - When indicating the logical value of a signal line: 1 and 0 are used, with 1 meaning assert and 0 meaning negate.
  - When indicating the actual value on a signal line: "H" and "L" are used, with "H" meaning high and "L" meaning low signal potential.
- Hexadecimal notation in these specifications is indicated by H'## (e.g., H'FF, H'00).

3. Interface Specifications

3.1 Interface Signals

The STbus basic interface signals are listed in Table 1, as seen from one function unit.

In this table, RQL*, RQH*, GR*, and ET* are signals connected individually to each function unit.

Table 1 - Basic Interface Signals (function unit interfaces other than bus handler)

No.	Signal name	Count	Functional category	Connection type
1	RQL∗ (Request low)	1	Arbitration control	Individually connected
2	RQH∗ (Request high)	1		
3	GR∗ (Grant)	1		
4	ET∗ (End of bus transaction)	1		
5	BS∗ (Bus transaction start)	1	Transfer control	Bus connection
6	BUR∗ (Burst)	1		
7	CSP∗ (Control signal parity)	1		
8	LCK∗ (Lock)	1		
9	AD [00..63]∗ (Command/address/data)	64	Command/address/data	
10	ADP [0..7]∗ (AD parity)	8		
11	RTY∗ (Retry)	1	Cache coherency control	
12	RST∗ (Reset)	1	Reset signal	
13	CK (Clock)	1	Clock	See Note 2.
	Total number of signals	83		

Note 1: A ∗ after a signal name indicates negative logic.

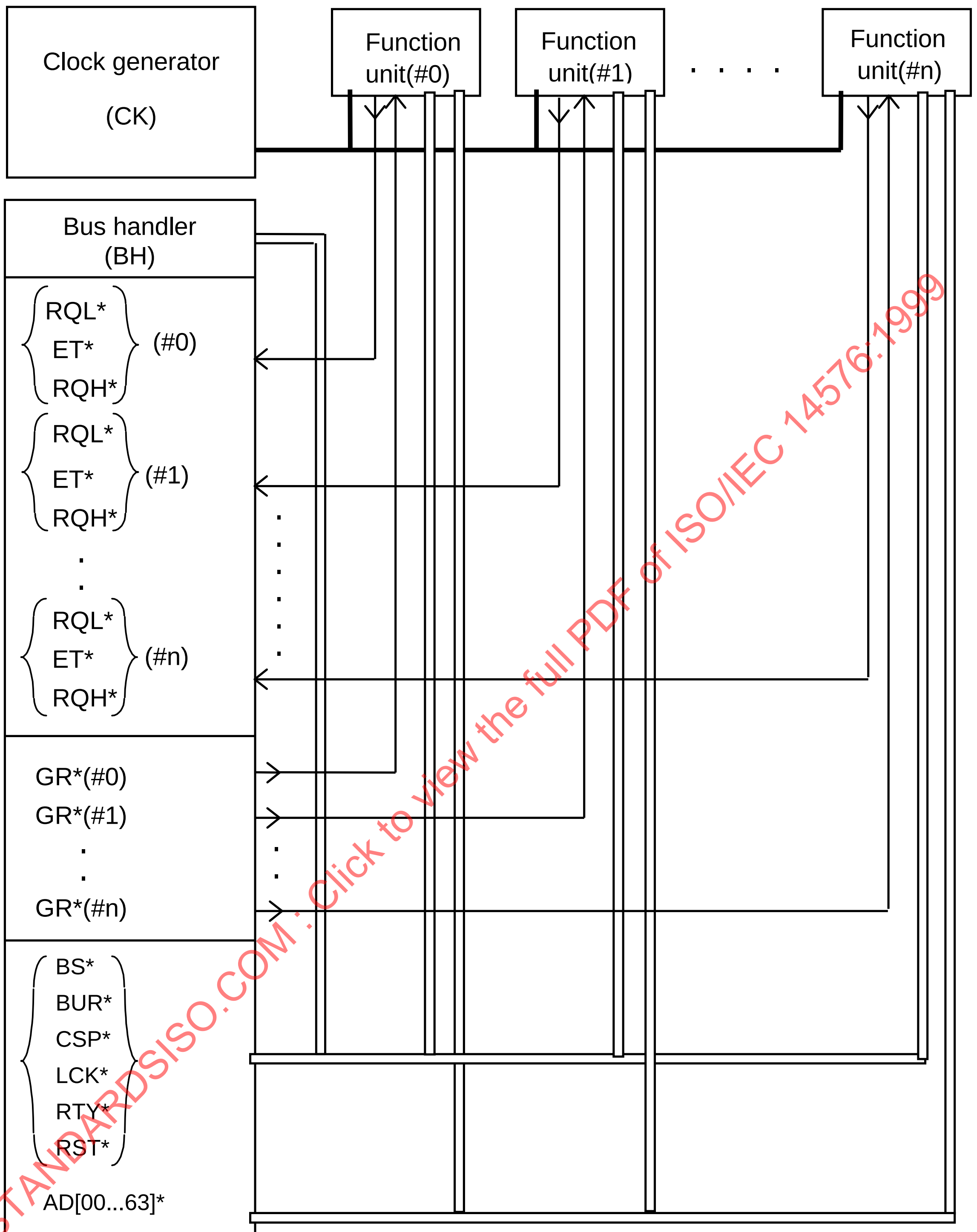
Note 2: For clock connection, a connection configuration must be adopted that can guarantee the skew specified in the physical specifications.

The optional interface signal lines as seen from one function unit are listed in Table 2. Since these signals are optional, the system implementor can choose whether or not to use them.

Table 2 - Optional Interface Signals (function unit interfaces other than bus handler)

No.	Signal name	Count	Functional category	Connection type
14	LCKS* (Lock spare)	1	Transfer control	Bus connection
15	RTYS* (Retry spare)	1	Cache coherency control	
16	STI* (Steal inhibit)	1		
17	STIS* (Steal inhibit spare)	1		

Connection structure



n+1: Connected function unit count
CK signal: Individual signal or bus signal
For clock connection, a connection configuration must be adopted that can guarantee the skew specified in the physical specifications.

Figure 2 - Connection interface between function units (basic pattern)

Explanation of each signal

1) RQL* (Request low)

This signal is used by a source unit (SUT) to request the bus. Each unit asserts this signal when performing an order transaction, for requesting the bus right from the bus handler. A function unit for which the GR∗ signal is asserted, granting the right to use the bus, must negate this signal.

This signal is notified to the bus handler by each function unit using individual lines.

This signal has a lower priority than that of RQH∗, RQL∗ and RQH∗ cannot be asserted simultaneously.

2) RQH* (Request high)

This signal is used by a destination unit (DUT) to request the bus. Each unit asserts this signal when performing an answer transaction, in order to request the bus right from the bus handler. A function unit for which the GR∗ signal is asserted, granting the right to use the bus, must negate this signal.

This signal is notified to the bus handler by each function unit using individual lines.

This signal has a higher priority than that of RQL∗, RQH∗ and RQL∗ cannot be asserted simultaneously.

While a LCK∗ signal is asserted, the bus handler will not assert GR∗ in response to a RQL∗ signal from another function unit. However, GR∗ will be asserted in response to RQH∗, so any unit is capable of executing an answer transaction.

3) GR* (Grant)

This signal is for granting the bus right to a bus master in response to a RQL∗ or RQH∗ bus request signal. Only while this signal is asserted, a function unit enables bus drivers (Nos. 5 - 7, 9,10, in Table 1) and send information on the bus. This signal is supplied to each function unit by the bus handler on individual lines.

4) ET* (End of bus transaction)

This signal is issued by the bus master to give advance notice of the end of transfer data.

This signal is negated two cycles prior to the actual end of a data transfer.

If this signal is not asserted at the same time as RQL∗ or RQH∗, this is taken to mean that the requested transaction is a one-cycle transfer.

For a transfer of two cycles or more, ET∗ is asserted at the same time as RQL∗ or RQH∗.

This signal is notified to the bus handler by each function unit using individual lines.

5) BS* (Bus transaction start)

When a function unit that has obtained the bus right performs an order transaction or answer transaction, this signal is asserted at the same time as the command or answer information is sent on the bus, indicating to the destination function unit the start of transfer information.

This signal is asserted only during the first bus cycle of an order transaction or answer transaction.

When data is sent following the initial command or answer information, information receipt must be performed at the initiative of the receiving function unit, using the BS∗ signal as a reference.

6) BUR* (Burst)

This signal indicates burst transfer mode, consisting of two or more data transfer cycles. The sending function unit asserts this signal, and while it is asserted the receiving function unit continues to receive data.

This signal is negated one cycle prior to the end of data transfer. When the receiving function unit detects the negation of this signal, it ends the receiving operation after one cycle.

If a BS∗ signal is asserted and BUR∗ is not asserted, one-cycle transfer is indicated.

7) CSP* (Control signal parity)

This is a parity signal for the transfer control signals BS∗ and BUR∗. It indicates odd parity.

8) LCK* (Lock)

This is a bus lock signal. It is asserted at the same time as the SUT starts an order transaction, and is negated when the SUT itself indicates the end of a transaction.

Split transfer is the main method adopted for STbus, but interlock transfer using this signal is also possible.

9) AD [00..63]* (Command / address / data 00-63)

This is a 64-bit two-way information transfer bus for time division transfer of control information, address information, and data.

10) ADP [0..7]* (AD parity 0-7)

These are parity signals for each byte of AD[00-63]∗. They indicate odd parity.

11) RTY* (Retry)

This signal is used for coherency control when a copyback cache scheme is used in a TCMP system. When this signal is asserted, the bus master must retry the current transaction. See 5.4 Retry Indication for details.

12) RST* (Reset)

A function unit connected to STbus uses this signal to indicate a reset to other function units. While this signal is asserted, reset is in effect. Assertion time of this signal is specified to be 5 µs-10 µs. Assertion timing shall be synchronized with the bus clock.

13) CK (Clock)

This is the STbus common clock signal. Bus operations are synchronized with the falling edge of this signal.

14) LCKS*(Lock spare)

This is a spare LCK* signal. When this signal is used, LCK* must also be used at the same time. If either LCK* or this signal is asserted alone, the lock transfer is not effective. The lock transfer is performed, only when both LCK* and this signal are asserted. This signal is optional.

15) RTYS* (Retry spare)

This is a spare RTY∗ signal. When this signal is used, RTY∗ must also be used at the same time. If either RTY∗ or this signal is asserted alone, the bus master must retry the current transaction. This signal is optional.

16) STI* (Steal inhibit)

This is a signal used in copyback cache coherency control. It is connected only between function units. This signal is optional.

When this signal is asserted, the bus master prohibits a steal operation during copyback of data concerned. See 5.5 Steal Operation for details.

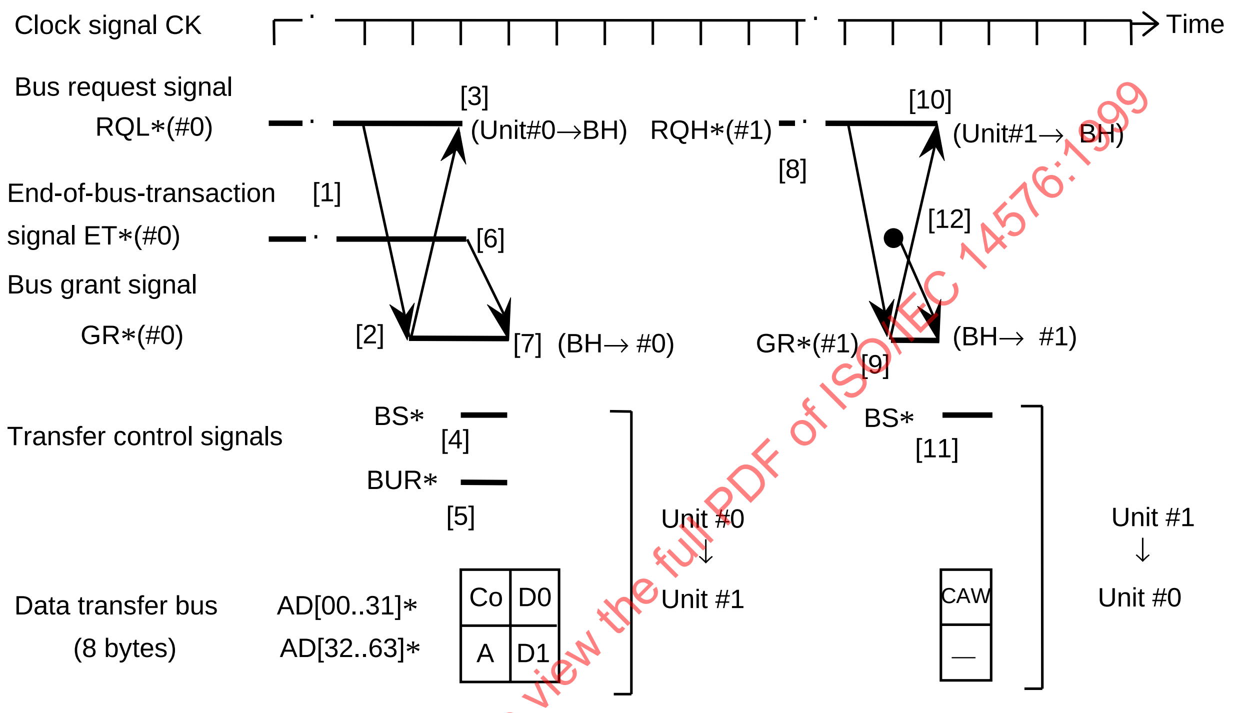
17) STIS* (Steal inhibit spare)

This is a spare STI∗ signal. When this signal is used, STI∗ must also be used at the same time. If either STI∗ or this signal is asserted alone, the bus master prohibits a steal operation during copyback of data concerned. This signal is optional.

4. Bus Operations

4.1 Protocol for Basic Operations

Examples of basic STbus operations (one-cycle transfer and two-cycle transfer) are given here, along with an explanation of the operations.



Abbreviations:
Co: Order command Di: Data A: Address CAW: Answer command BH: Bus handler
[1]-[12]: Change point of each signal line (numbers correspond to explanation below)

Figure 3 - Concept of bus operation protocol (for 1-cycle or 2-cycle transfer: 8-byte bus width specification, write operation)

All function units and the bus handler operate in synchronization with the common clock CK.

[1] In the figure above, function unit (#0) performs a two-word information transfer to function unit (#1). First, function unit (#0) asserts a bus request signal RQL∗(#0), requesting the bus handler to grant the bus right. At the same time, since the number of words to be transferred is two, it also asserts an end-of-bus-transaction signal ET∗(#0).

[2] If the RQL∗(#0) signal can be accepted, the bus handler asserts a bus grant signal GR∗(#0) to function unit (#0), granting it the bus right.

[3] Function unit (#0), obtaining the bus right, negates the RQL∗(#0) signal upon the assertion of GR∗(#0).

[4] At the same time as the above, function unit (#0) upon receiving GR∗(#0) validates the information transfer bus and sends information on the bus. In the initial cycle of information transfer, it asserts a BS∗ transfer control signal.

[5] Since in this example a two-word information transfer is performed, a BUR∗ signal is asserted at the same time as BS∗, informing the other function units of the start of burst transfer. The BUR∗ signal is negated one cycle prior to the end of the transaction.

[6] The ET∗(#0) asserted at the same time as GR∗(#0) is negated two cycles prior to the end of the transaction, notifying the bus handler in advance that the bus is about to be freed.

[7] When the bus handler detects the negation of ET∗(#0) it negates GR∗(#0).

[8] When answer information (one word) is returned in response to the information from function unit (#0), function unit (#1) asserts a bus request signal RQH∗(#1), requesting the bus handler to grant the bus right. As long as the number of words to be transferred in the answer is one, ET∗(#1) is not asserted.

[9] If the RQH∗(#1) signal can be accepted, the bus handler asserts a bus grant signal GR∗(#1) to function unit (#1).

[10] Function unit (#1), obtaining the bus right, negates the RQH∗(#1) signal upon the assertion of GR∗(#1).

[11] At the same time as the above, function unit (#1) upon receiving GR∗(#1) validates the information transfer bus and sends information on the bus. In the initial cycle of information transfer, it asserts a BS∗ transfer control signal. Since the number of words to be transferred in the answer is one, BUR∗(#1) is not asserted.

[12] Since ET∗(#1) is negated, the bus handler negates GR∗(#1) in one-word transfer.

Since a split transfer method is adopted, the bus can be used for transactions between other function units during steps [7] to [9] above.

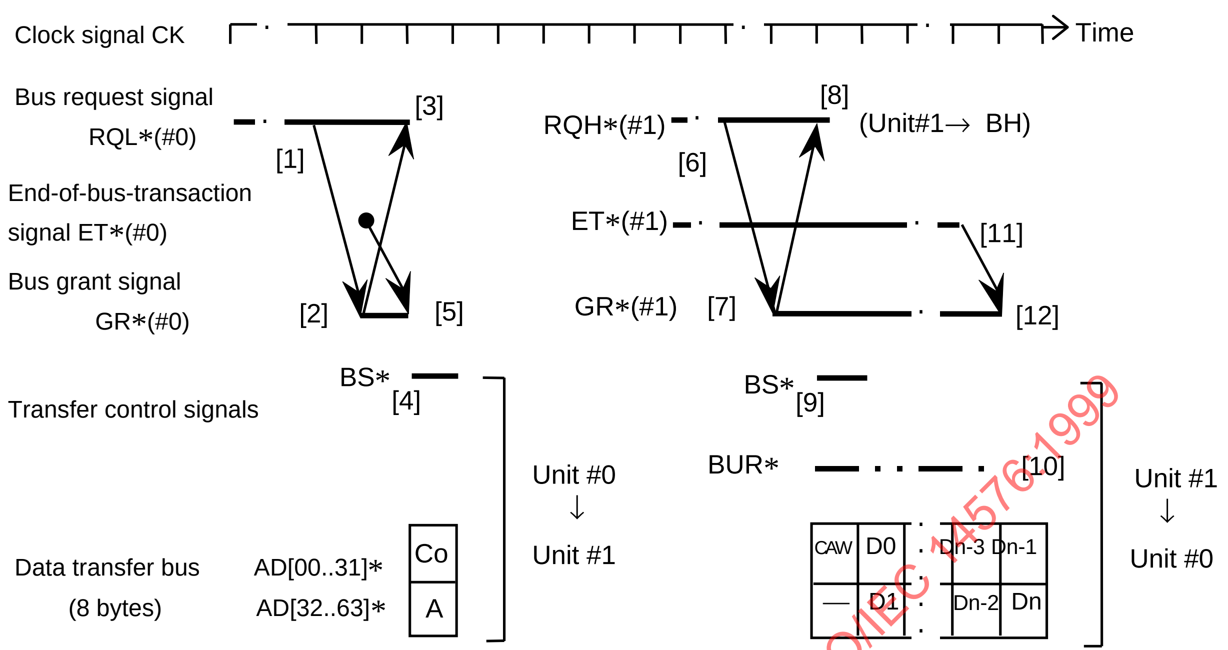


Figure 4 – Concept of bus operation protocol (for transfer of 3-cycles or more: 8-byte bus width specification, read operation)

[1] The figure above shows one-word information transfer between function unit (#0) and function unit (#1). First, function unit (#0) asserts a bus request signal RQL∗(#0), requesting the bus handler to grant the bus right. Since the number of words to be transferred is one, ET∗(#0) is not asserted.

[2] If the RQL∗(#0) signal can be accepted, the bus handler asserts a bus grant signal GR∗(#0) to function unit (#0), granting it the bus right.

[3] Function unit (#0), obtaining the bus right, negates the RQL∗(#0) signal upon the assertion of GR∗(#0).

[4] At the same time as the above, function unit (#0) upon receiving GR∗(#0) validates the information transfer bus and sends information on the bus. In the initial cycle of information transfer, it asserts a BS∗ transfer control signal. Since the number of words to be transferred in this example is one, BUR∗(#0) is not asserted.

[5] Since RQL∗(#0) and ET∗(#0) are negated, the bus handler negates GR∗(#0) in one-word transfer.

[6] When answer information of three words or more is returned in response to the information from function unit (#0), function unit (#1) asserts a bus request signal RQH*(#1), requesting the bus handler to grant the bus right. In a multi-word answer, ET*(#1) must be asserted at the same time.

[7] If the RQH∗(#1) signal can be accepted, the bus handler asserts a bus grant signal GR∗(#1) to function unit (#1).

[8] Function unit (#1), obtaining the bus right, negates the RQH∗(#1) signal upon the assertion of GR∗(#1).

[9] At the same time as the above, function unit (#1) upon receiving GR∗(#1) validates the information transfer bus and sends information on the bus. In the initial cycle of information transfer, it asserts a BS∗ transfer control signal.

[10] Since in this example three words or more of information are transferred, a BUR∗ signal is asserted at the same time as BS∗, informing the other function unit of the start of burst transfer. The BUR∗ signal is negated one cycle prior to the end of the transaction.

[11] The ET∗(#1) asserted at the same time as GR∗(#1) is negated two cycles prior to the end of the transaction, notifying the bus handler in advance that the bus is about to be freed.

[12] The bus handler monitors the bus for negation of the ET∗(#1) signal, and when this is detected it negates GR∗(#1).

The operation when a 4-byte bus width is used (one-cycle or two-cycle transfer) is shown below for reference.

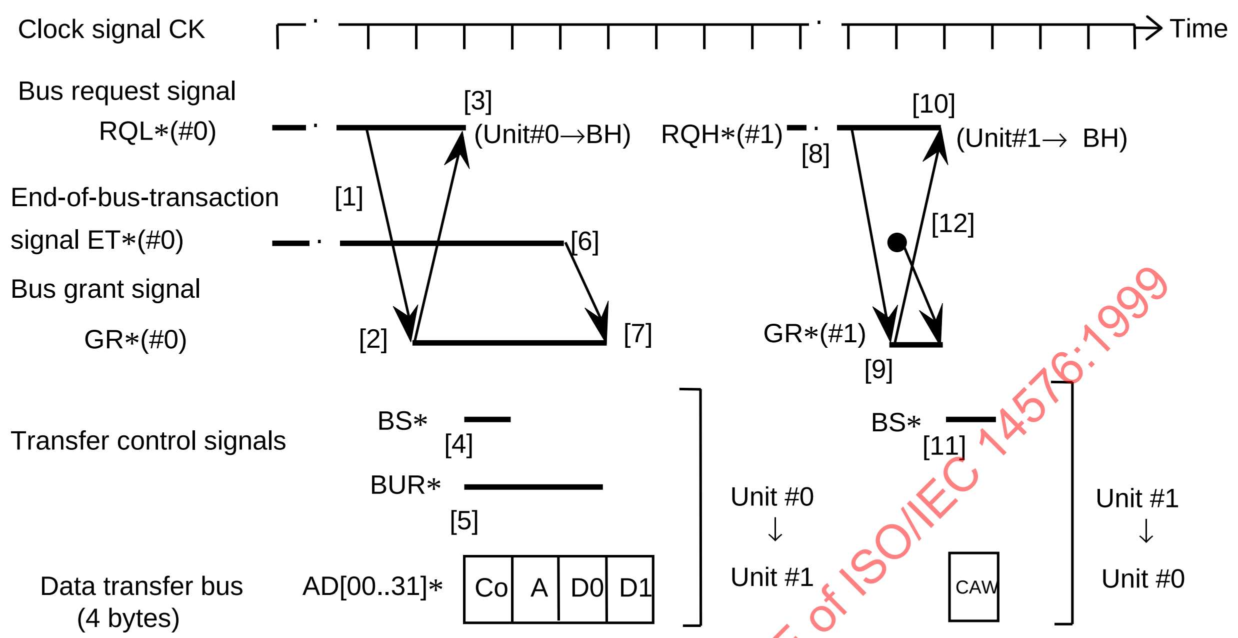


Figure 5 - Concept of bus operation protocol (for 1-cycle or 2-cycle transfer: 4-byte bus width specification, write operation)

When the width of the information bus is 4 bytes, the operation by which function unit (#0) sends two-word information to function unit (#1) is essentially the same as with an 8-byte bus. Since, however, the bus width is 4 bytes, the information that is sent on an 8-byte bus in one cycle requires two cycles on a 4-byte bus, as shown in the figure above.

The explanation of each signal change point is the same as for the 8-byte specification.

4.2 Transfer Protocol

4.2.1 Bus operation types

The bus operations on STbus can be classified broadly into four types; memory access, control space access, message transfer and control register access.

1) Memory access

   This is a bus operation in which a function unit performs a read or write operation in shared memory or in the local memory of any other function unit. In the basic bus specification any transfer byte size can be designated up to the maximum of 256 bytes.

2) Control space access

   This is a bus operation in which a program running on a function unit directly reads or writes data mapped to the control space of any other function unit. In the basic bus specification any transfer unit byte size can be designated up to the maximum of 256 bytes.

3) Message transfer

   This is a bus operation in which a function unit sends a message to any other function unit.

4) Control register access

   This is a bus operation in which a program running on a function unit directly reads or writes up to 8 bytes in the control registers (up to 256) of any other function unit.

4.2.2 Command format

The information transfer bus is 64 bits wide (or 32 bits), and adopts a big endian. The bit order is AD [00..63]∗ (or AD [00..31]∗) starting from the MSB.

The information transfer bus is divided into byte units, in the order from byte 0 to byte 7 starting from the MSB.

Information transferred on the bus can be classified broadly into the four types commands, addresses, data, and message communication operands.

In information transfer, the commands are sent first to the destination unit. The format is shown in Table 3. Although not indicated specifically in the table, a single parity bit is attached to the above byte units.

Table 3 - Command Format for Information Transfer Bus

Bus Operation	Byte 0		Byte 1		Byte 2							Byte 3
	OPT 0		OPT 1		OPT 2							
	00	01.............07	08	09.............15	16	17	18	19	20	21	22 23	24.............31
Memory access	0	BMID	0	BSID	0	BT	R/W	A64	M	NAT	AID	BCT
Control space access	0	BMID	0	BSID	1	BT	R/W	A64	0	NAT	AID	BCT
Message transfer	0	BMID	1	BSID	0	BT	MD	SQ		NAT	AID	BCT
Control register access	0	BMID	1	BSID	1	BT	R/W	BCT			AID	RA
Reserved	1	BMID	0	BSID	0	BT	*	*	*	*	* *	* * * * * * * *
Reserved	1	BMID	0	BSID	1	BT	*	*	*	*	* *	* * * * * * * *
Reserved	1	BMID	1	BSID	0	BT	*	*	*	*	* *	* * * * * * * *
Answer	1	BMID	1	BSID	1	BT	ROPT			RNAT	RAID	ANS

Notation:

OPT: Operation Type	**BCT: Byte Count**
BMID: Bus Master ID	**MD: Mode**
BSID: Bus Slave ID	**SQ: Sequence**
BT: Bus Type	**RA: Register Address**
R/W: Read/Write	**ROPT: Return Operation Type**
A64: 64-bit Address Space	**RNAT: Return NAT**
M: Modify	**RAID: Return Access ID**
NAT: No-Answer Transaction	**ANS: Answer code**
AID: Access ID	
	***: Reserved**

1) Operation type codes

Commands are either for order transactions, from SUT to DUT, or for answer transactions, from DUT to SUT. There are eight command types, including Reserved. Together these are called operation types, and are defined in a 3-bit OPT (operation type) consisting of bit 0 of byte 0, bit 8 of byte 1, and bit 16 of byte 2.

The OPT codes are given in Table 4.

Table 4 - OPT Code Definitions

OPT 0 1 2	**Operation type**
0 0 0	Memory Access
0 0 1	Control Space Access
0 1 0	Message Transfer
0 1 1	Control Register Access
1 0 0	Reserved
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Answer

2) ID (Identifier) and Bus Type (BT)

In an order transaction, a BMID (Bus Master Identifier) and BSID (Bus Slave Identifier) are sent in bytes 0 and 1 of the information transfer bus.

Bit 17 of byte 2 indicates the BT (Bus Type).

The semantics of each field are as follows.

BMID (Bus Master Identifier):
    Function unit ID of the bus master, consisting of 7 bits.

BSID (Bus Slave Identifier):
    Function unit ID of the bus slave, consisting of 7 bits.

BT (Bus Type):
    0: 4-byte bus   1: 8-byte bus

3) Memory access

The semantics of each field are as follows.

R/W (Read/Write):

0: Write operation by SUT to DUT
1: Read operation by SUT from DUT

A64 (64-bit Address Space)

0: 32-bit address space
1: 64-bit address space

M (Modify):

This signal is used in cache coherency control in a TCMP configuration. It is used in combination with R/W, as specified in Table 5.

Table 5 - M Bit Definition

R/W	M	Bus operation
0	0	Memory Write Operation
	1	Cache Invalidation
1	0	Memory Read Operation
	1	Memory Read Operation and Cache Invalidation

When the R/W field is 0 and the M field is 1, the BCT field is invalid, so that only commands and addresses can be transferred on the bus. Write operations to memory are not performed. If a cache hit occurs in another function unit, the cache is invalidated.

Note: If copyback cache is not supported, the M field is cleared to 0.

AID (Access Identifier):

An identifier used when multiple access is made to memory. Applications include simultaneous operation of multiple DMA ports, or pipeline operation of a single DMA port.
An example of multiple memory access operation in a split transfer bus is shown in Figure 6 below.

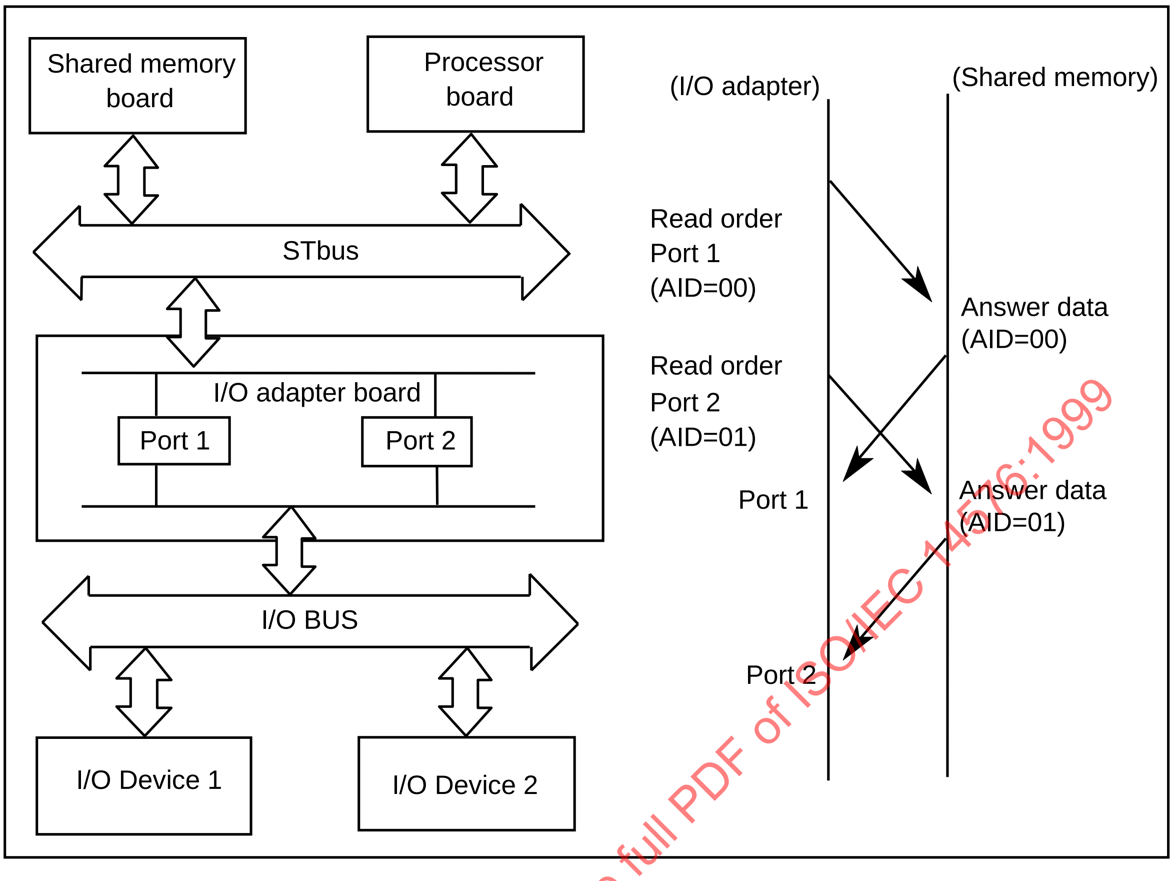


Figure 6 - Pipeline operation

NAT (No-Answer Transaction):

This indicates that the transaction to follow is a no-answer transaction. When this bit is set in an order transaction, the bus slave does not perform an answer transaction in response. However, if an error occurs in the order transaction, an answer to indicate the error is returned.

0: Transaction with answer returned

1: Transaction without answer returned

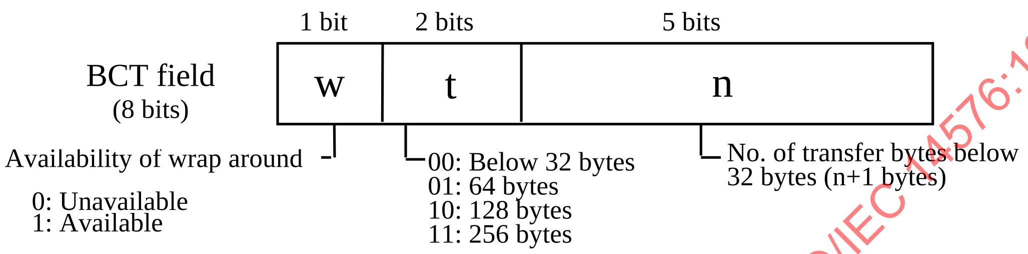
BCT (Byte Count):

This indicates the number of bytes of transfer data sent from SUT to DUT or demanded from DUT by SUT. The number of transfer bytes are defined as shown in Figure 7 a). Up to 256 bytes can be sent consecutively in one bus operation. When t="00", the number of valid bytes transferred on the bus must equal the contents of n plus 1. For example, t= "00" & n="00000" means 1-byte transfer, while t="00" & n="11111" means 32 bytes transfer.

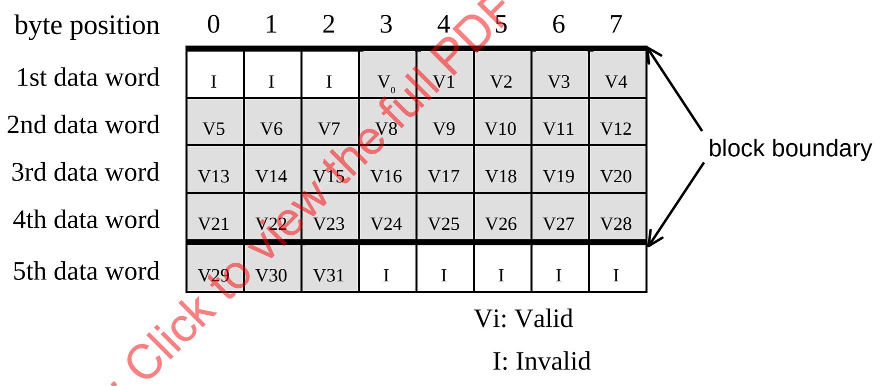
Byte alignment in case of an 8-byte bus is realized using the lower 3 bits of the address and the BCT contents. Starting from the byte position of the first word of data as determined by the lower 3 address bits, the number of bytes as determined from BCT is the valid data. If the address is 8n + 3 and BCT is t="00" & n="11111" & w="0" (32-byte transfer), the valid data is as shown in Figure 7 b) below.

On the various data sent in a transfer operation, a correct parity must be attached even to invalid data.

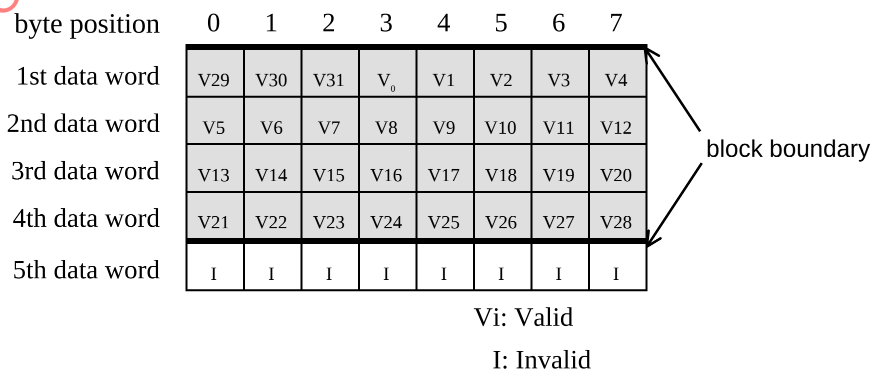
Note, however, that in case of w="1" (used on block size transfer in general) the arrangement of data is wrapped around at the block boundary. When the address is 8n+3 and BCT is t="00" & n="11111" & w="1", the valid data is as shown in Figure 7 (c) below.



a) BCT field definition


byte position
0 1 2 3 4 5 6 7
1st data word: I I I V0 V1 V2 V3 V4
2nd data word: V5 V6 V7 V8 V9 V10 V11 V12
3rd data word: V13 V14 V15 V16 V17 V18 V19 V20
4th data word: V21 V22 V23 V24 V25 V26 V27 V28
5th data word: V29 V30 V31 I I I I I
block boundary
Vi: Valid
I: Invalid

b) BCT and byte alignment in case of w="0"


byte position
0 1 2 3 4 5 6 7
1st data word: V29 V30 V31 V0 V1 V2 V3 V4
2nd data word: V5 V6 V7 V8 V9 V10 V11 V12
3rd data word: V13 V14 V15 V16 V17 V18 V19 V20
4th data word: V21 V22 V23 V24 V25 V26 V27 V28
5th data word: I I I I I I I I
block boundary
Vi: Valid
I: Invalid

c) BCT and byte alignment in case of w="1"

Figure 7 - BCT field

4) Control space access

The semantics of each field are as given below.

R/W (Read/Write): same as memory access.

A64 (64-bit Address Space): same as memory access.

AID (Access Identifier): same as memory access.

NAT (No-Answer Transaction): same as memory access.

BCT (Byte Count): same as memory access (including the relation between BCT and address).

5) Message transfer

The semantics of each field are as given below.

MD (Mode): Indicates the difference in message processing urgency.
0: urgent message 1: normal message

SQ (Sequence): The message semantics are specified in Table 6 below.

Table 6 - Message sequence

SQ		**Meaning**	**Explanation**
20	21		
0	0	Single	Indicates that the message length is 256 bytes or less and the transaction ends with one bus operation. The receiving unit upon accepting this bus operation notifies the processor that message receipt is complete.
0	1	First	Indicates the first bus operation when the message length exceeds 256 bytes. The receiving unit state is bus operation receive pending state.
1	0	Middle	Indicates a bus operation continuing after a First or other Middle operation. When Middle is received, the receiving unit state is next message receive pending state. When the bus operation is received, the receiving unit first confirms that BMID and AID are the same, then accepts the bus operation and waits for the next bus operation.
1	1	Last	A bus operation continuing after a First or Middle operation. The operation on the receiving end is the same as when Middle is received, but since the last message is indicated, after it is received the receiving unit notifies the processor that message receipt is complete.

NAT (No-Answer Transaction): same as memory access.

BCT (Byte Count): same as memory access. Note, however, that since no address is sent with this command, the valid data is that indicated by BCT starting from byte 0 of the first data word.

Note on message transfer parameters:

The message transfer parameters are 8 bytes of additional information sent following the message communication commands. The operand contents include such items as the subsystem ID in communication between subsystems. Detailed contents are left up to the implementor.

6) Control register access commands

The semantics of each field are as given below.

R/W (Read/Write): same as memory access.

BCT (Byte Count):

Designates the number of bytes of data to be transferred. This field consists of 3 bits, so up to 8 bytes can be designated per bus operation.
The number of valid bytes transferred on the bus must equal the BCT contents plus 1. For example, BCT = H'7 means 8-byte transfer.

RA (Register Address):

Designates the register address. The byte alignment is defined by the lower 3 bits of RA and the BCT contents. Starting from the byte position of the first word of data as determined by the lower 3 RA bits, the number of bytes as determined from BCT is the valid data. If the address is 8n + 3 and BCT is X'6' (7-byte transfer), the valid data is as shown in Figure 8 below.

byte position	0	1	2	3	4	5	6	7
1st data word	I	I	I	V0	V1	V2	V3	V4
2nd data word	V5	V6	I	I	I	I	I	I

Vi: Valid

I: Invalid

Figure 8 - RA and byte alignment

7) Answer

The answer command in an answer transaction is the first information returned by DUT to SUT. The pairing of BMID and BSID is the same as in an order transaction; but since the bus master is a DUT, BMID is the DUT ID and BSID is the SUT ID.

The semantics of each field are as given below.

ROPT (Return Operation Type):
The OPT code sent by SUT to DUT in the order transaction is embedded in this field.

RNAT(Return NAT):
The NAT designated in the order transaction is embedded in this field.

RAID (Return Access Identifier):
The AID designated in the order transaction is embedded in this field.

ANS (Answer code):
Answer information, consisting of 8 bits and indicating the results of the request made in the order transaction. Part of the field is used for status information at the bus slave. Definition of the remaining contents is left to the implementor.

4.2.3 Transfer sequence

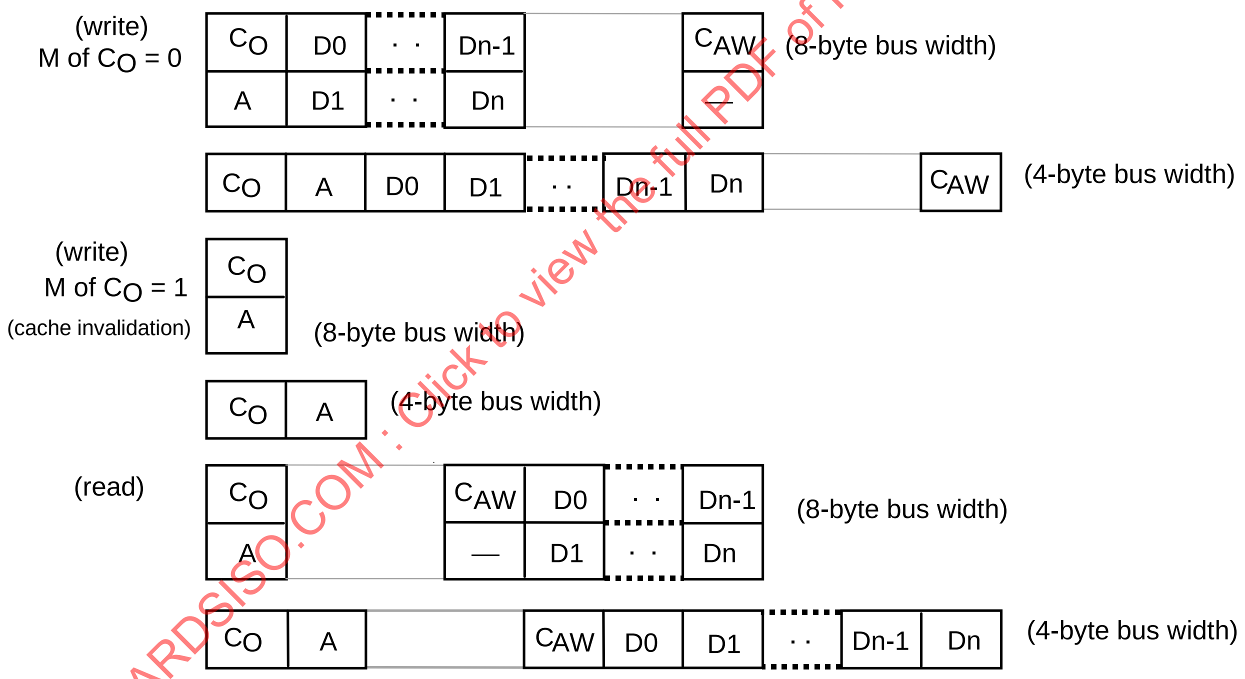
The transfer sequence is fixed, and the bus operation sequence depends on the type of command.

The notation used in the sequence diagrams below is as follows.

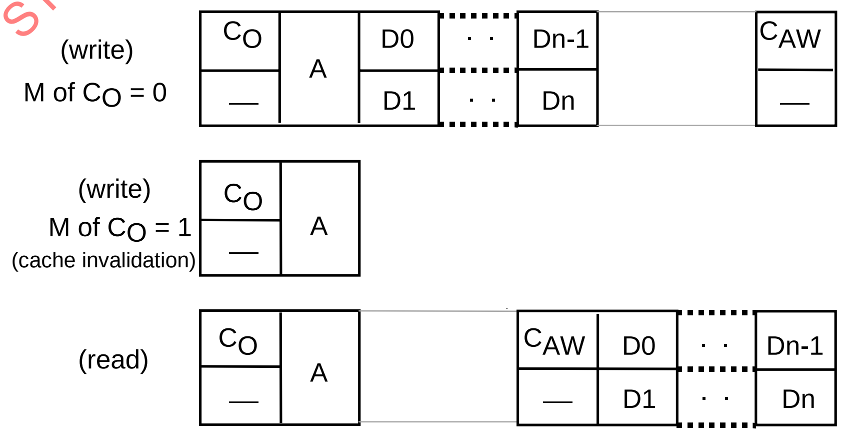
C_O:	Order command (in order transaction)
Di:	Data
C_{AW}:	Answer command (in answer transaction)
A:	Address
P:	Parameter

1) Memory access

a) 32-bit addressing

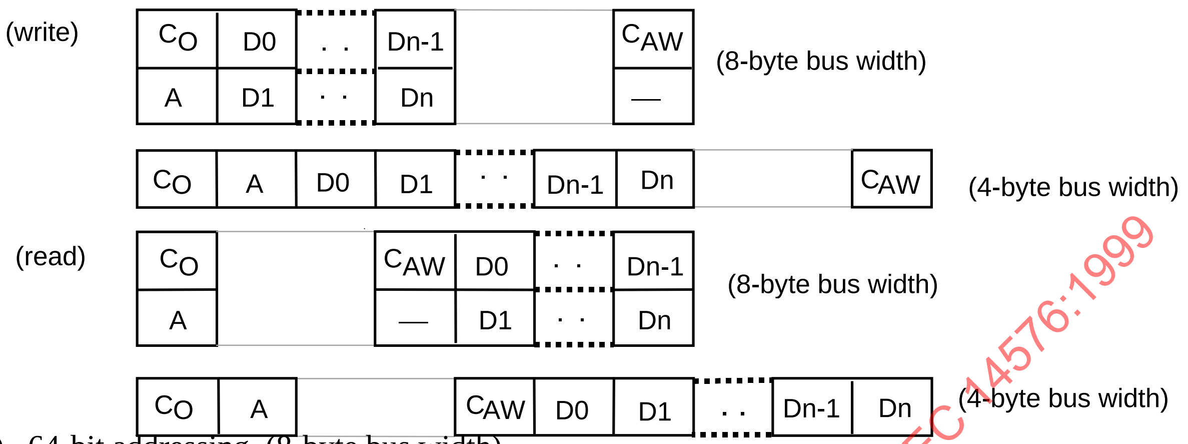


b) 64-bit addressing (8-byte bus width)

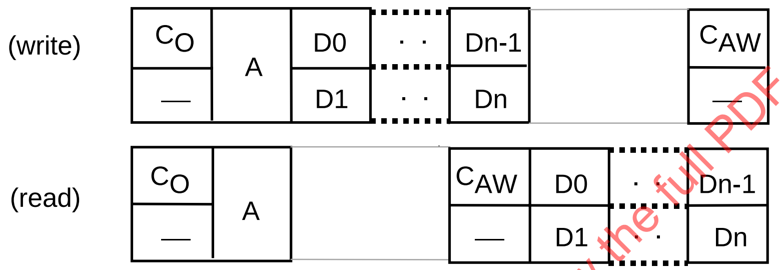


2) Control space access

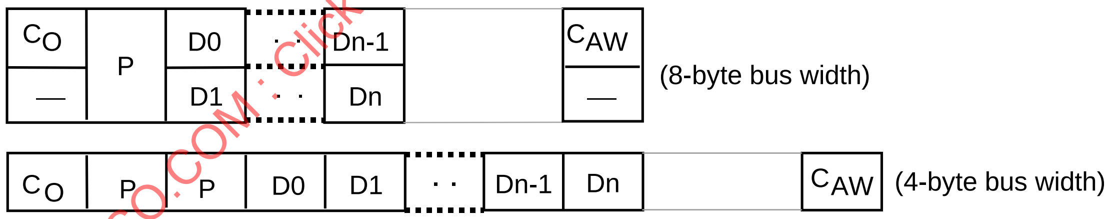
a) 32-bit addressing



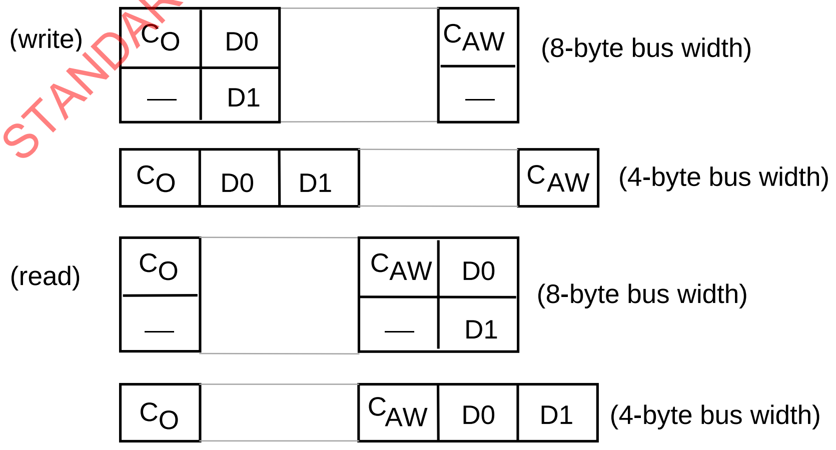
b) 64-bit addressing (8-byte bus width)



3) Message transfer



4) Control register access



4.3 Arbitration

STbus arbitration is centralized arbitration by means of the bus handler, which is configured as indicated in 3.1 Connection Interface. Requests are made to the bus handler from each unit via two kinds of bus request lines. The bus handler issues bus grant signals in response to these requests.

STbus does not specify the algorithm for priority control.

4.4 Status Reports

This section specifies the information on "transfer data receive status at the bus slave side" reported in answer transactions.

A status report by DUT to SUT in an answer transaction is for status indication in response to an order transaction. The answer codes are defined as shown in Table 7, using bits in the answer fields.

Table 7 - Answer Code Definition

Answer code values								Semantics	Classification
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**		
0	0	0	0	0	0	0	0	No error	Answer code defined by system manager (in case of no error)
		0	0	0	0	0	1	No error (Lock transfer)	
		0	0	0	0	1	0	Reserved	
		⋮						⋮	
		1	1	1	1	1	1	Reserved	
	1	0	0	0	0	0	0	User dependent	Answer code defined by user (in case of no error)
		⋮						⋮	
		1	1	1	1	1	1	User dependent	
1	0	0	0	0	0	0	0	Hardware error	Error report defined by system manager
		0	0	0	0	0	1	Illegal command	
		0	0	0	0	1	0	Bus sequence error	
		0	0	0	0	1	1	Reserved	
		⋮						⋮	
		1	1	1	1	1	1	Reserved	
	1	0	0	0	0	0	0	User dependent	Error report defined by user
		⋮						⋮	
		1	1	1	1	1	1	User dependent	

1) No error: This indicates that an order transaction has been received correctly by DUT.

2) No error (Lock transfer): In a lock transfer, this indicates that an order transaction has been received correctly by DUT.

3) Hardware error: This indicates that a bus signal parity error has been detected at DUT.

4) Illegal command: This indicates the receipt of a bus command that cannot be accepted. This is returned when, for example, a non-supported OPT or BT is received.

5) Bus sequence error: This indicates that DUT has detected a discrepancy between the data size sent by SUT and the transfer data size designated in the order command, or indicates that BS* has been asserted more than one cycle.

4.5 Data Transfer

This section specifies details of each bus operation.

A memory access write operation is explained in greatest detail as an example, while the other operations are given in outline. The operation sequences given in this section are for 32-bit addressing. On the sequences for 64-bit addressing, refer to the operation sequences in this section plus the transfer sequences in 4.2.3.

Only normal operation sequences are explained here. On sequences when error occurs during a transaction, see 4.8 Error Handling.

4.5.1 Memory access (write)

The bus sequences for a write operation in memory access are shown in Figures 9 a) - d). Of these, a) and c) are for one-word memory write and n-word memory write in a no-answer transaction, while b) and d) are for one-word memory write and n-word memory write in a basic-format transaction.

All system bus operations are synchronized with the falling edge of the bus clock (the vertical dotted lines in the figures). The figures take into account gate delay, accounting for the deviation from the bus clock line in the case of some signals.

Note that the part of the figure to the right of "Bus" indicates the timing for information occurring on the bus between function units.

The explanation below deals mainly with Figure 9 d) on n-word memory write (basic operation format).

[1] When a new memory access request occurs, function unit (#0) asserts RQL∗(#0), issuing a bus request signal.

[2] If the RQL∗(#0) signal can be accepted, the bus handler asserts a bus grant signal GR∗(#0) to function unit (#0), granting it the bus right.

[3] Function unit (#0) negates the RQL∗(#0) signal upon the assertion of GR∗(#0).

[4] Function unit (#0) obtaining the bus right by means of GR∗(#0) next validates the AD bus for two-way information transfer, and asserts BUR∗ and CSP∗ signals, then sends information on the bus in the sequence C_O (command) and A (address), then Di (data).

[5] At the same time as the first information is sent, function unit (#0) asserts a BS∗ (bus transaction start) signal declaring to the other function units the start of the necessary bus operation.

[6] Function unit (#0) negates the ET∗ signal two cycles prior to the end of the transfer data, declaring the end of the transaction to the bus handler.

[7] Function unit (#0) negates the BUR∗ (burst) signal one cycle prior to the end of the transfer data, declaring the end of the transaction to the DUT (destination function unit).

[8] When the bus handler detects the negation of ET∗(#0) it negates GR∗(#0), revoking the bus right of function unit (#0).

[9] Function unit (#1) prepares a normal C_{AW} (answer) for notifying function unit (#0) of normal write completion, and asserts RQH∗(#1), requesting the bus right. If error is detected, an error answer is prepared and RQH∗(#1) is asserted.

[10] If RQH∗(#1) can be accepted, the bus handler asserts a bus grant signal GR∗(#1), granting the bus right to function unit (#1).

[11] Function unit (#1) negates RQH∗(#1) upon the assertion of GR∗(#1).

[12] Function unit (#1), obtaining the bus right by means of GR∗(#1), validates the AD bus, then sends the C_{AW} on the bus.

[13] Function unit (#1), at the same time as it sends C_{AW}, asserts BS∗ with BUR∗ remaining negated, declaring the start and end of the transaction.

[14] The bus handler upon detecting the negation of ET∗ (#1) negates GR*(#1), revoking the bus right of function unit (#1).

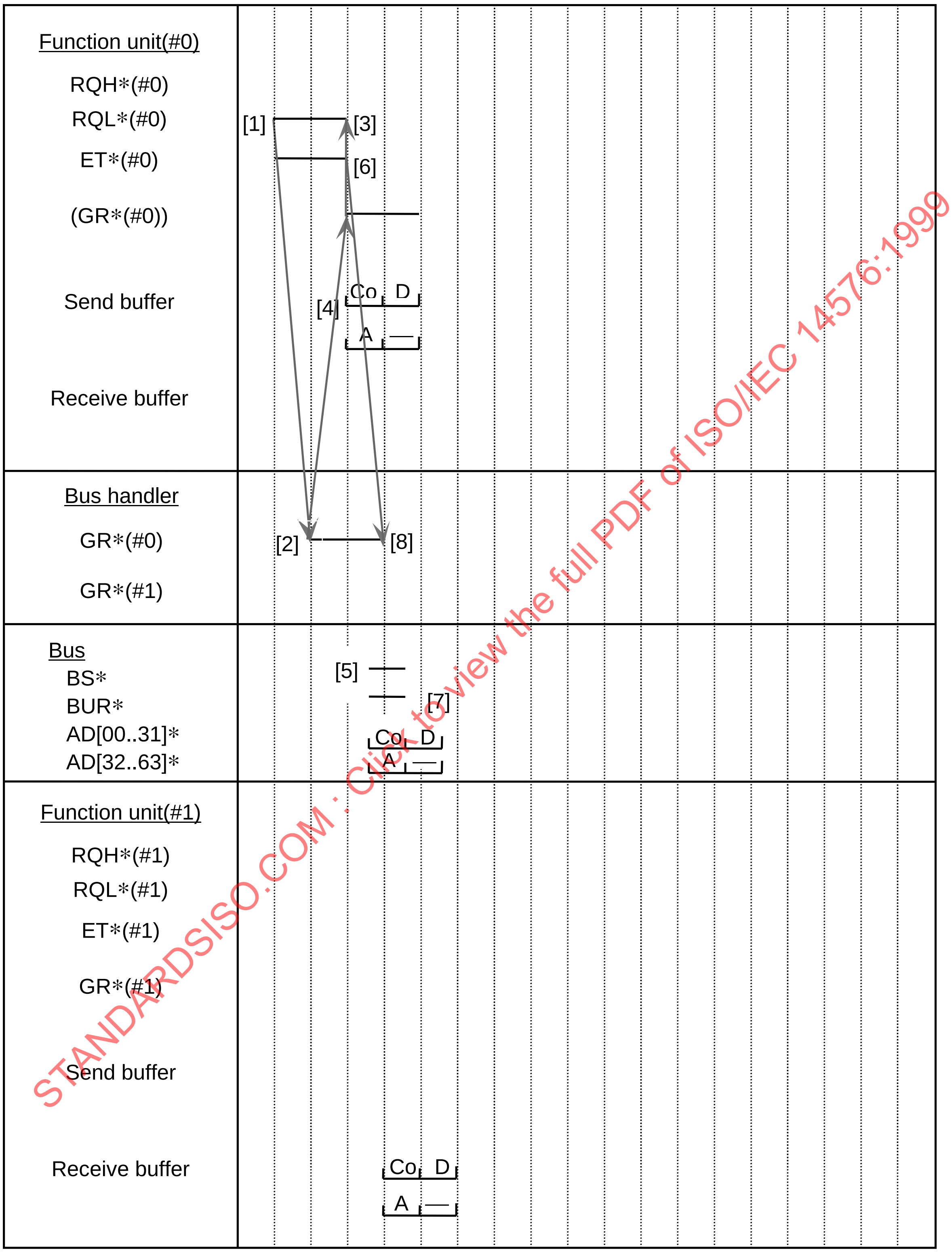


Figure 9 a) - One-word memory write (no-answer transaction)

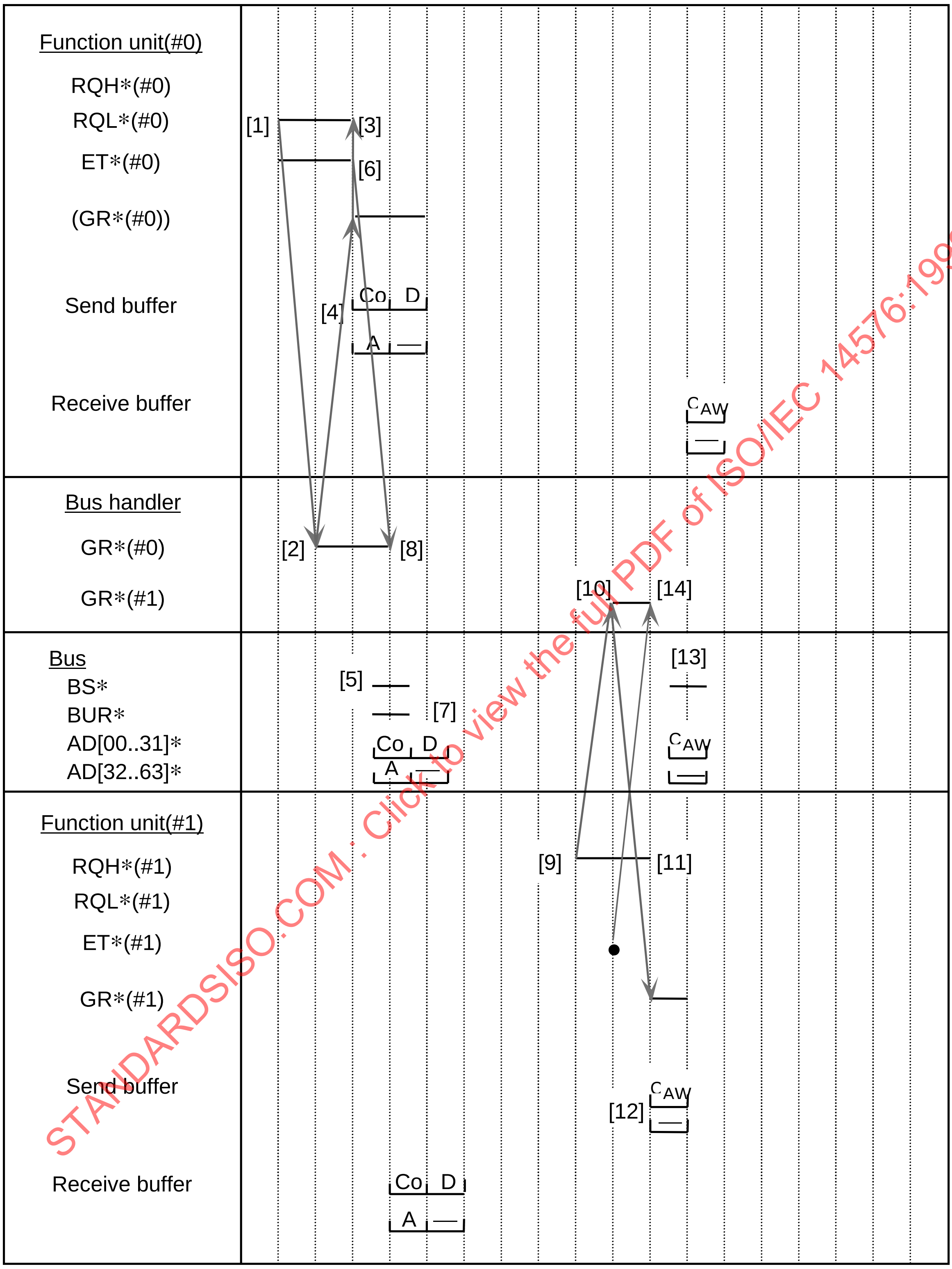


Figure 9 b) - One-word memory write (basic transaction)

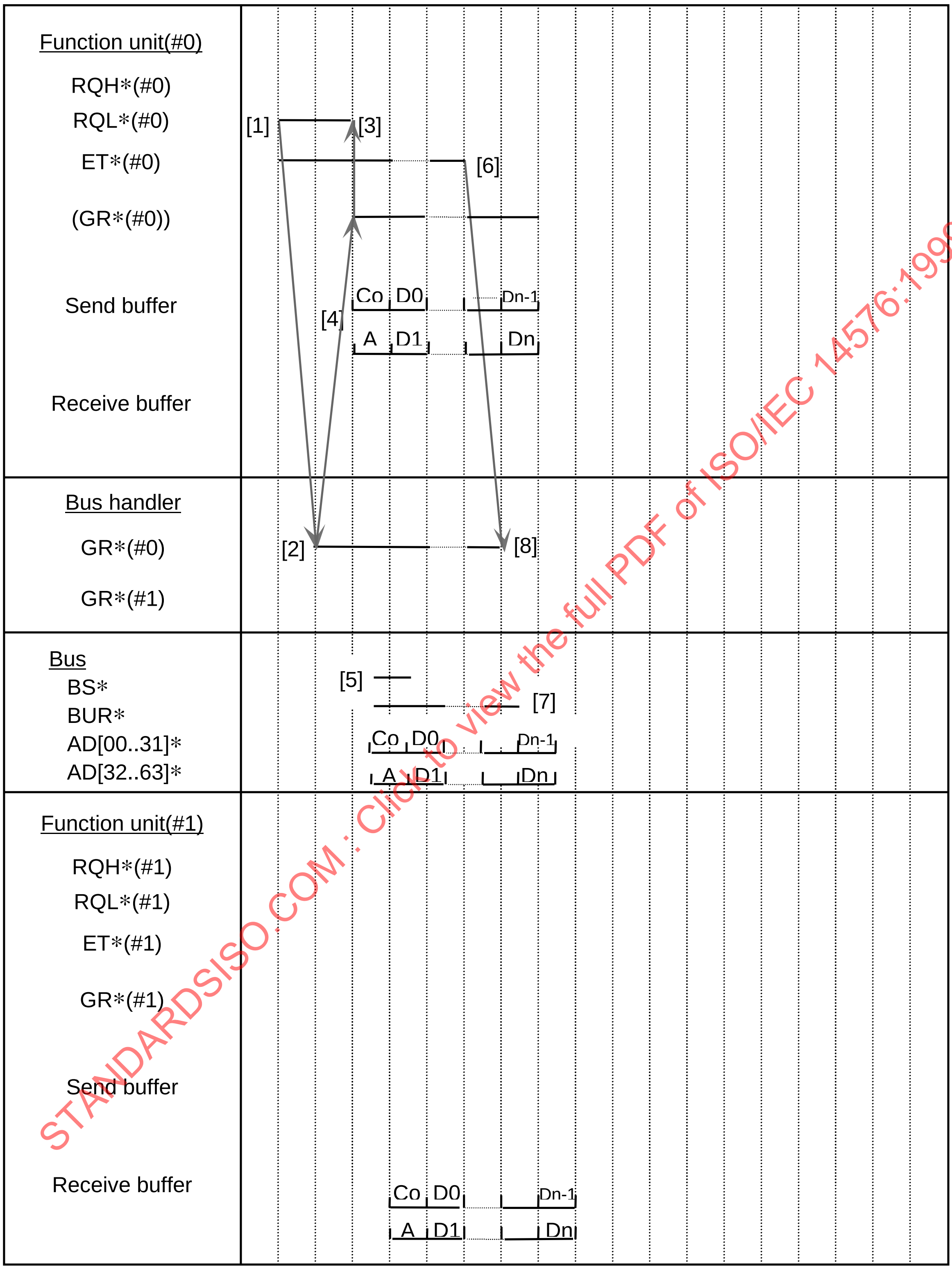


Figure 9 c) - n-word memory write (no-answer transaction)

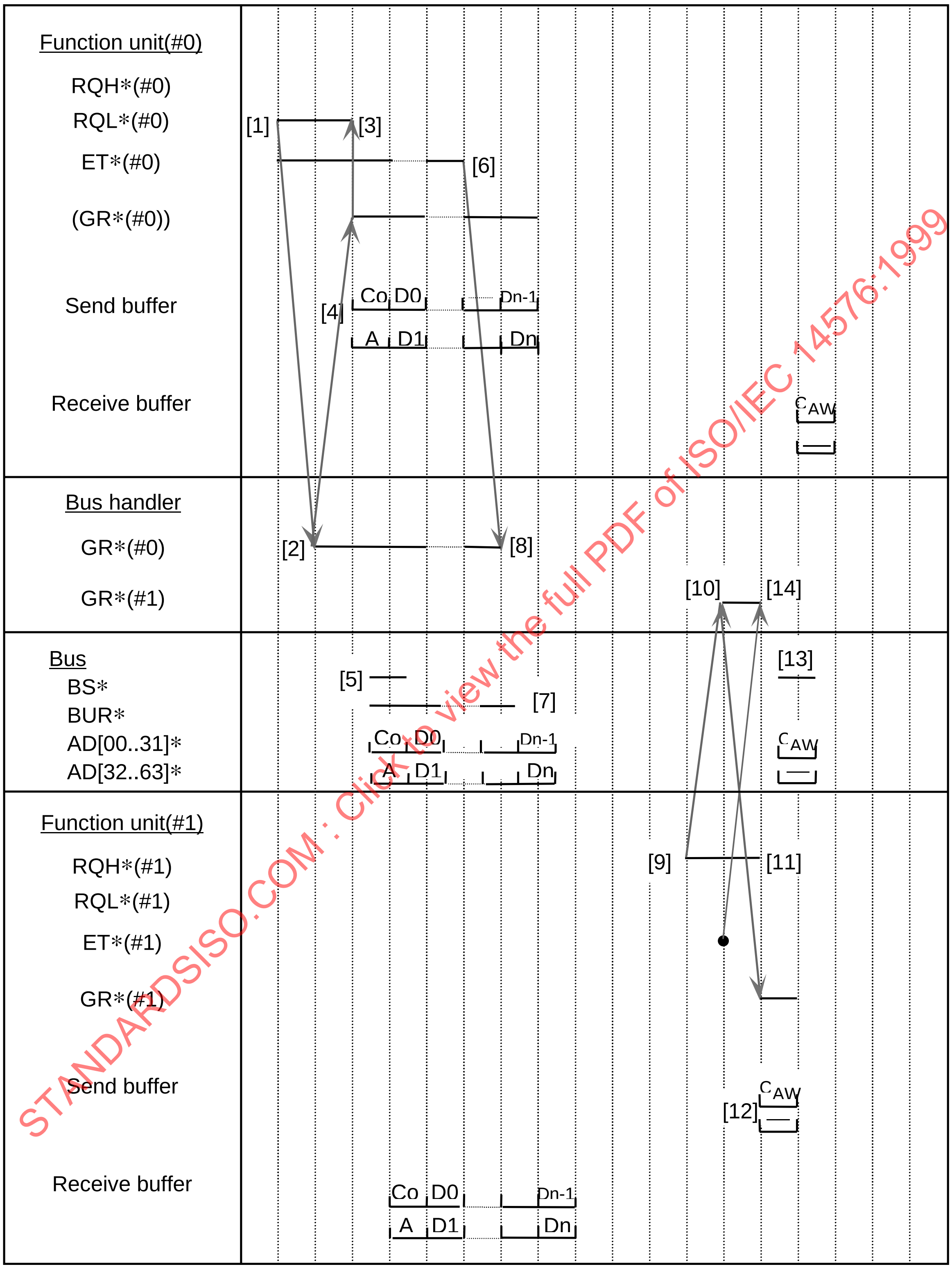


Figure 9 d) - n-word memory write (basic transaction)

4.5.2 Memory access (read)

The bus sequences for a read operation in memory access are shown in Figures 10 a) - b).

The function unit corresponding to DUT begins an operation for reading data from the memory area indicated by the address in its own local memory.

After the read operation is complete, DUT first sends a C_{AW} (answer) and then D (read data) to SUT.

SUT checks the C_{AW} contents and the data, and if these are normal, begins an operation for writing the read data to its own local memory. When the write operation is complete, the memory access read operation is ended.

If the C_{AW} contents contain error, or if error is detected in the SUT local memory write operation, this is notified to the software as a bus error in general.

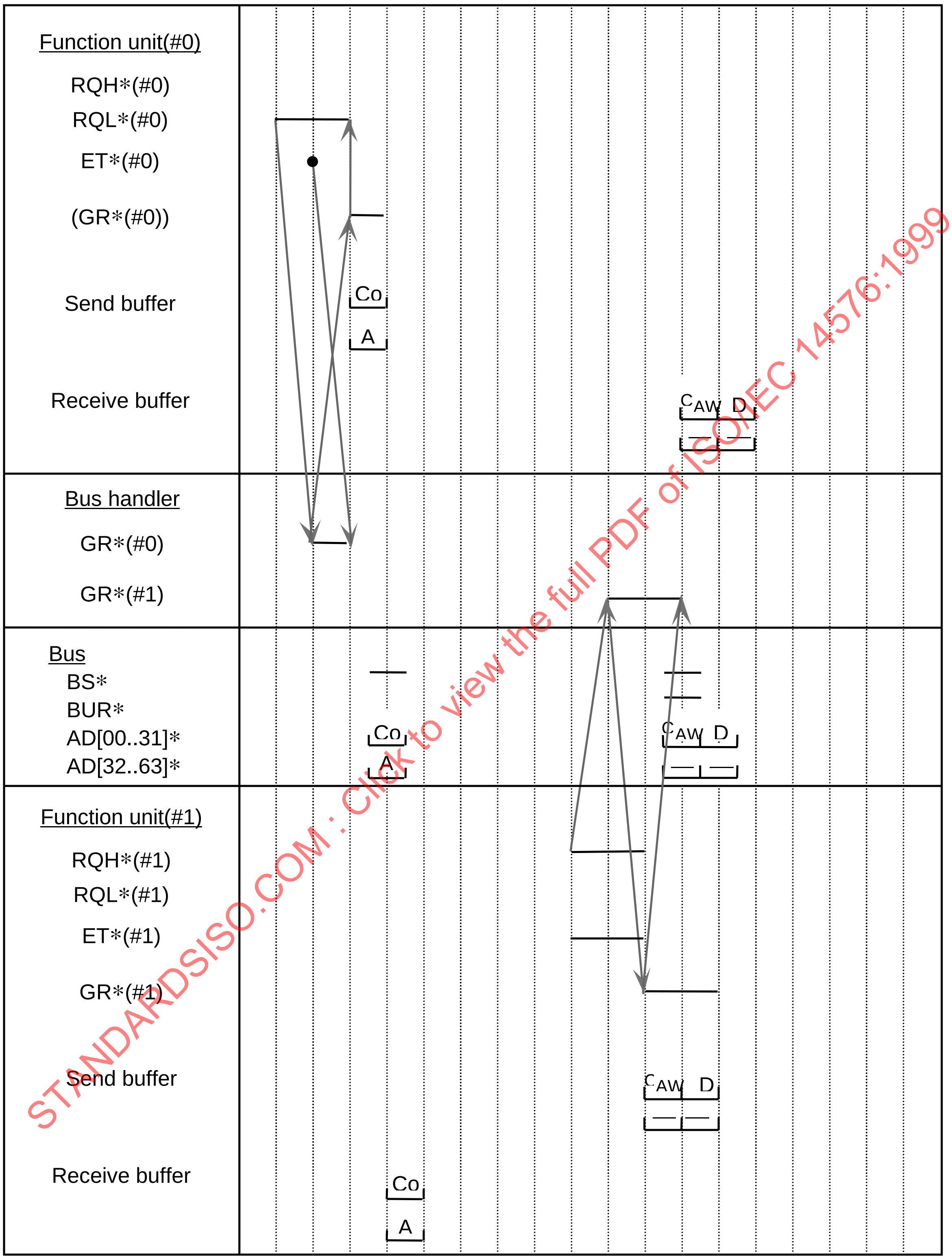


Figure 10 a) - One-word memory read

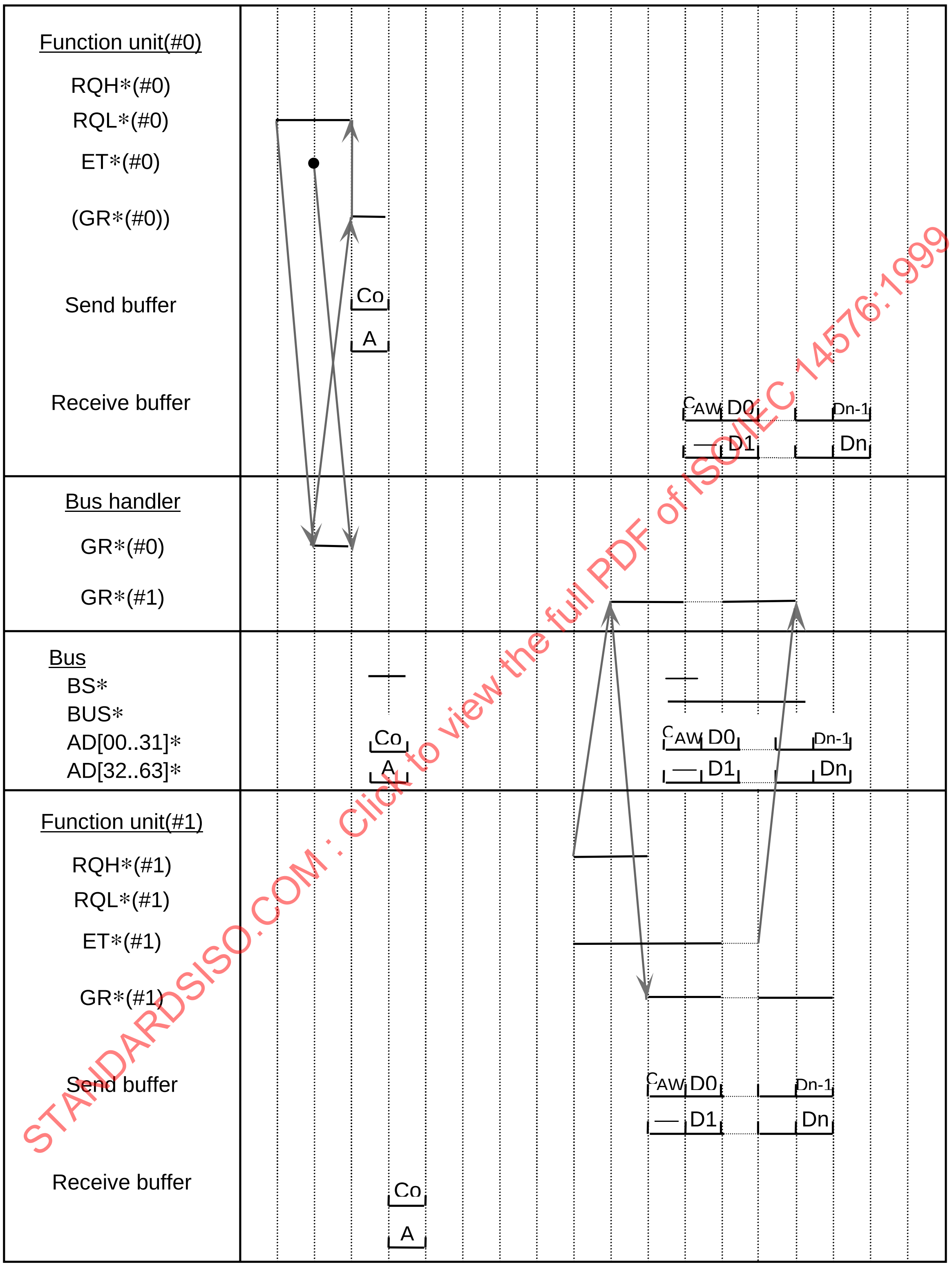


Figure 10 b) – n-word memory read

4.5.3 Control space access (write)

The bus sequence for a write operation in control space access is shown in Figure 11.

The function unit corresponding to DUT writes data to the control space indicated by the address in its own local memory.

After the operation is complete, DUT sends a C_{AW} (answer) for the operation to SUT. SUT checks the C_{AW} contents, and if normal, ends the write operation to the control space address.

If the C_{AW} contents contain error, this is notified to the software as a bus error.

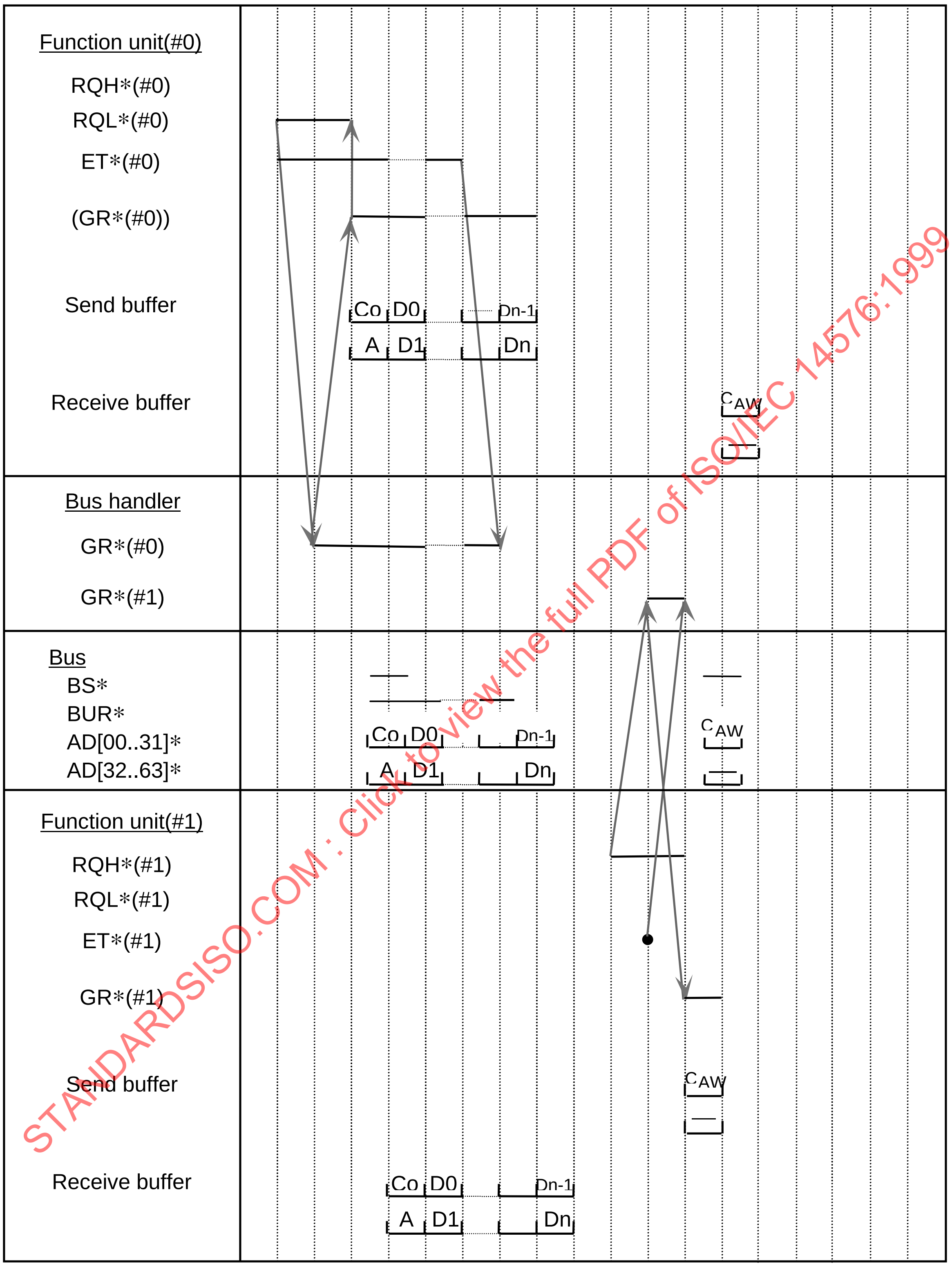


Figure 11 - n-word write: control space access

4.5.4 Control space access (read)

The bus sequence for a read operation in control space access is shown in Figure 12.

The function unit corresponding to DUT begins an operation for reading data from the control space area indicated by the address in its own local memory.

After the read operation is complete, DUT first sends a C_{AW} (answer) for the operation and then D (data) to SUT.

SUT checks the C_{AW} contents, and if these are normal, begins an operation for writing the read data to its own local memory.

If the C_{AW} contents contain error, or if error is detected in the SUT local memory write operation, this is notified to the software as a bus error.

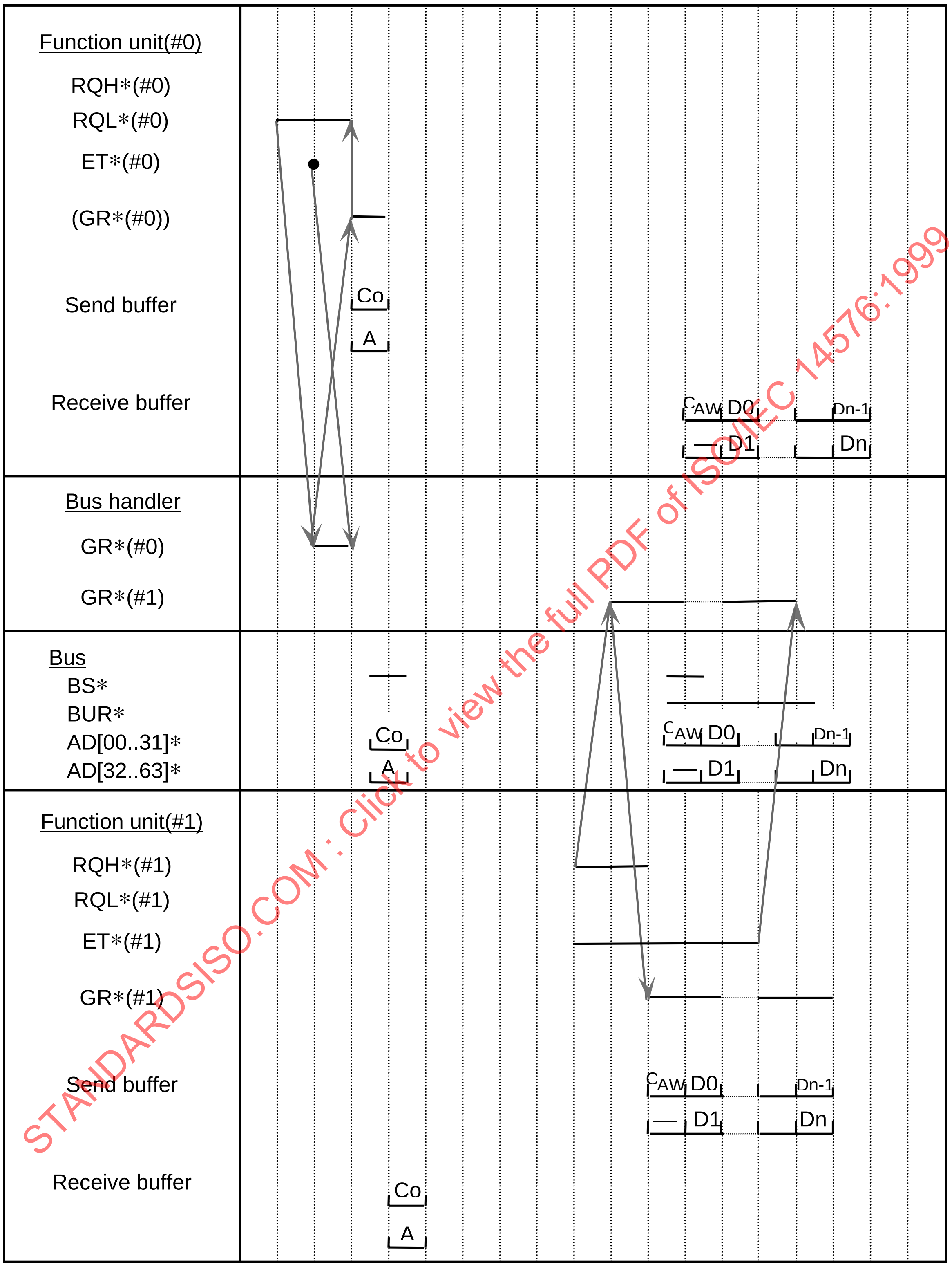


Figure 12 - n-word read: control space access

4.5.5 Message transfer

The bus sequence for a message transfer operation is shown in Figure 13.

The DUT designated by BSID in the C_O (command) begins a data write operation as instructed by MD.

After the write operation is complete, DUT returns a C_{AW} (answer) for the operation to SUT.

SUT checks the C_{AW} contents, and if these are normal, ends the message transfer operation.

If the C_{AW} contents contain error, this is notified to the software as a bus error.

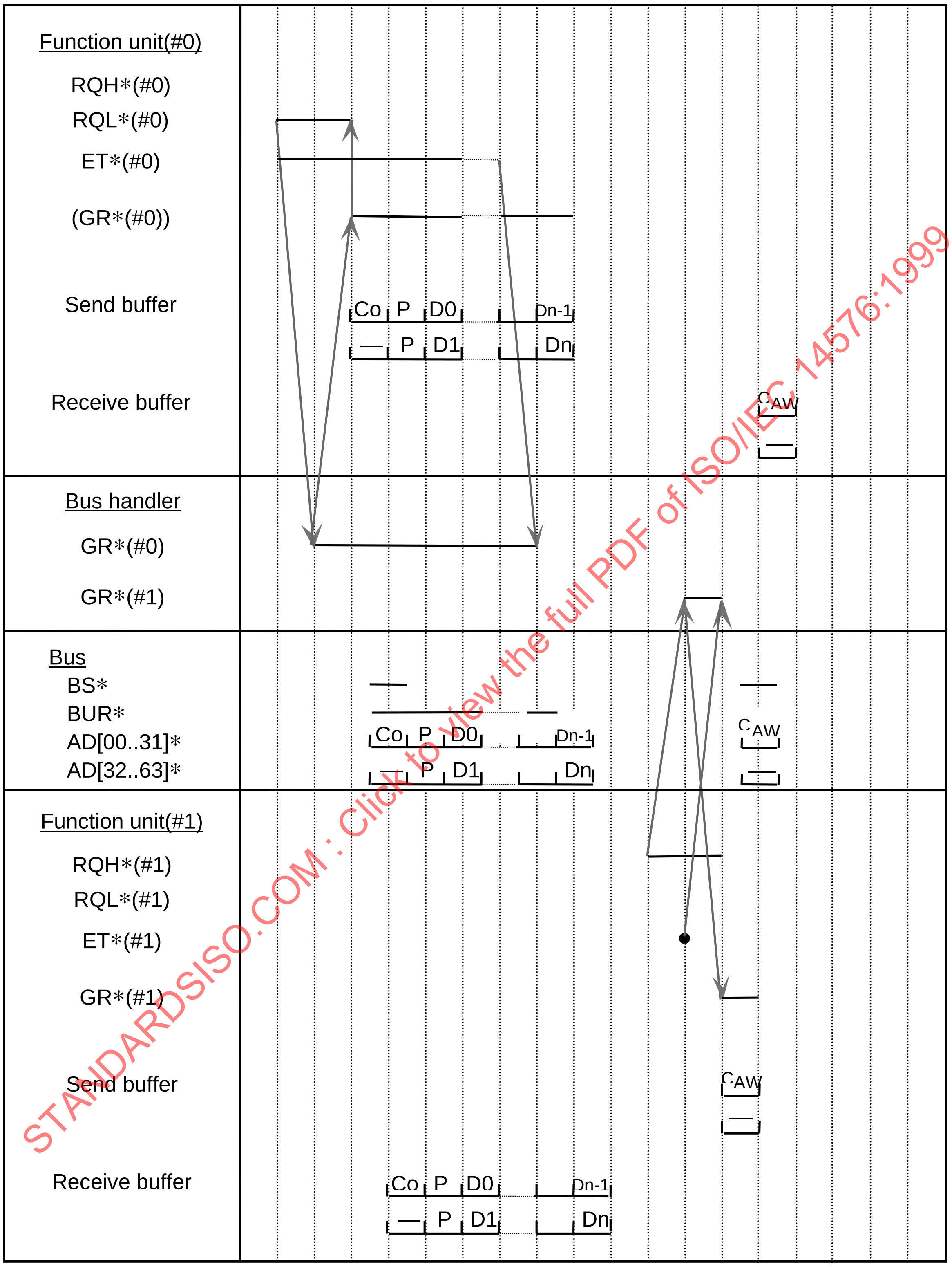


Figure 13 – n-word message transfer

4.5.6 Control register access (write)

The bus sequence for a write operation in control register access is shown in Figure 14.

The function unit corresponding to DUT writes data to the control register indicated by RA in the C_O (command).

After the operation is complete, DUT returns a C_{AW} (answer) for the operation to SUT.

SUT checks the C_{AW} contents, and if these are normal, ends the control register access write operation.

If the C_{AW} contents contain error, this is notified to the software as a bus error.

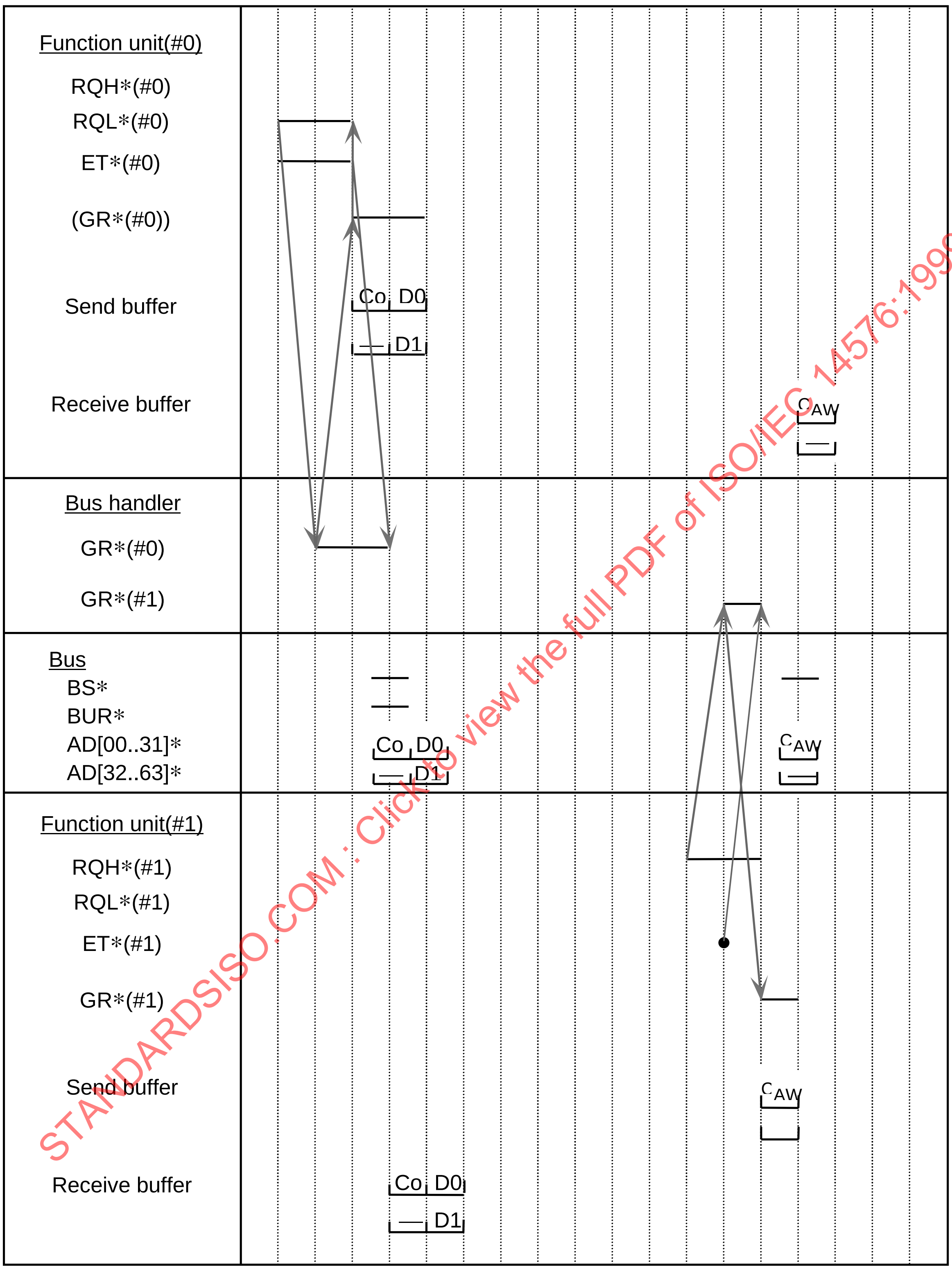


Figure 14 - One-word write: control register access

4.5.7 Control register access (read)

The bus sequence for a read operation in control register access is shown in Figure 15.

The function unit corresponding to DUT reads data from the control register indicated by RA in the C_O (command).

After the read operation is complete, DUT returns a C_{AW} (answer) for the operation.

SUT checks the C_{AW} contents, and if these are normal, latches the read data and ends the control register access read operation.

If the C_{AW} contents contain error, this is notified to the software as a bus error.

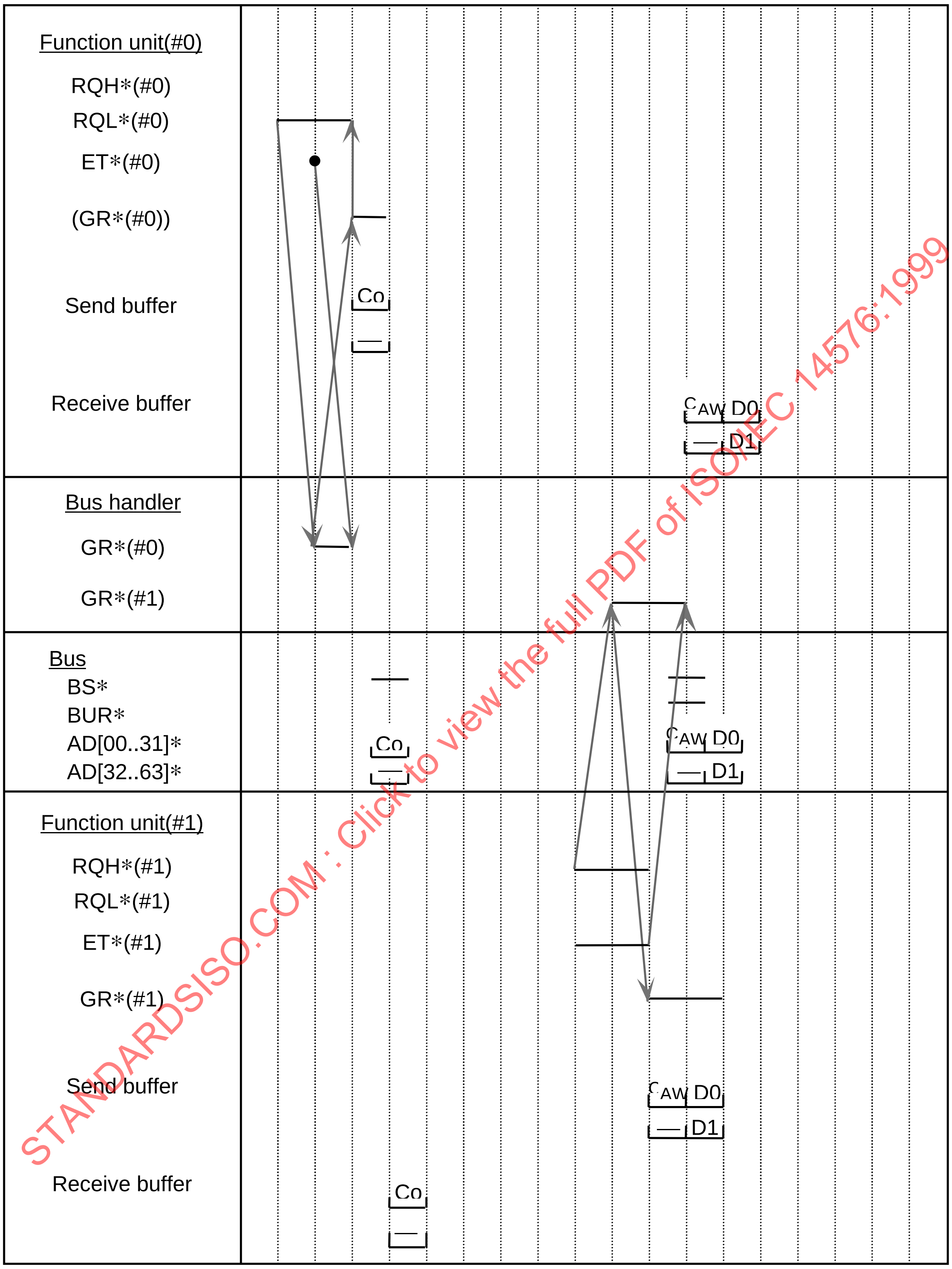


Figure 15 - One-word read: control register access

4.6 Lock Operations

STbus supports lock transfer operations, by which a function unit maintains exclusive right to the bus, for use in consecutive information transfer operations. For example, the lock transfer is used for a Test & Set operation (a modified read and write operation), etc.

The bus sequence for a lock transfer operation is shown in Figure 16. In the example shown in this figure, read and write bus operations take place consecutively during the time a LCK∗ signal is asserted.

SUT must assert both the RQL∗ signal and the ET∗ signal simultaneously when SUT requests the bus right for a lock transfer, even if the order transaction has one cycle transfer (See Figure 16).

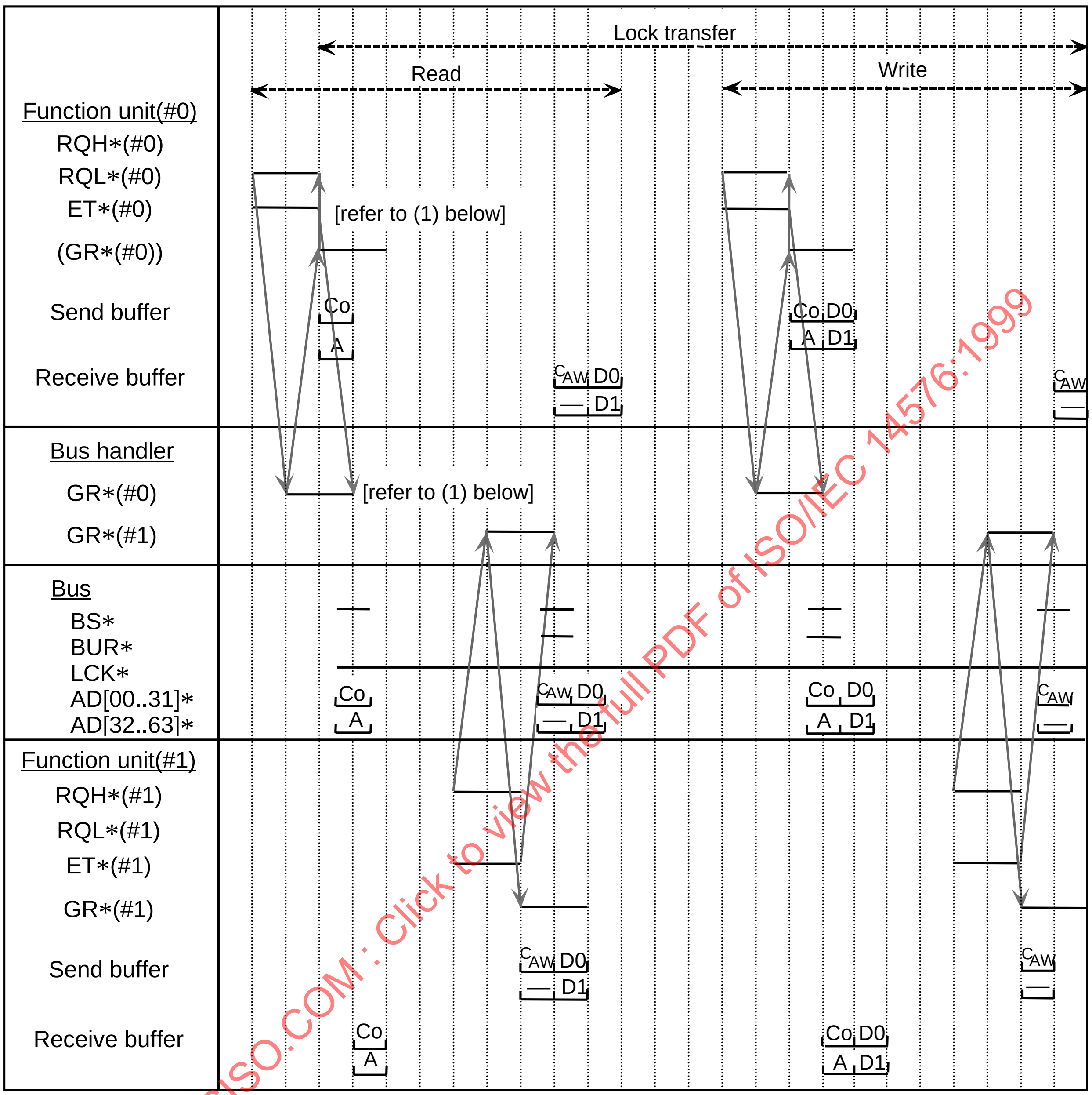
The LCK∗ signal is asserted by SUT at the same time as the BS∗ signal. The LCK∗ negate timing depends on the timing by which the processor ends the exclusive control instruction. Note, however, that the LCK∗ signal must continue to be asserted until SUT completes its last transaction. In this way exclusive right to the bus is guaranteed only during the time LCK∗ is asserted.

While the LCK∗ signal is asserted, the bus handler does not assert GR∗ in response to a RQL∗ request from another function unit.

GR∗ should be asserted, however, in response to a RQH∗ request, so any function unit can execute an answer transaction (See Annex B).

As means to notify the acknowledgment of the lock transfer on the reception side to the transmission side, a lock reception code should be provided to the answer codes (See Table 7).

Note that the way of lock transfer to cache data under the EM (Exclusive and Modified) state is shown in Annex C.



(1) Even if another FU#2 asserts RQL∗(#2), the bus handler should not assert GR∗(#2) to FU(#2) next bus cycle of assertion of GR∗(#0) to FU(#0).

However, the bus handler was too late to suspend GR∗(#2) assertion after recognition of the lock signal assertion.

For this reason, one cycle transfer in lock transfer operation is executed like two cycle transfer. That is, it should be specified to assert the ET∗signal with the RQL∗signal simultaneously.

Figure 16 - Bus lock transfer

4.7 Cache-related Operations

This section specifies system bus operations that support a copyback cache scheme.

Only the operation sequences on the system bus are described in this section. For cache state transitions after each bus operation, refer also to Chapter 5 on Cache Coherency Control.

As cache coherency support functions are optional, users may not necessarily implement the functions.

4.7.1 Cache invalidation

In processor bus interface control, when memory write commands are regularly monitored (bus snoop) and the write address is found in cache memory, cache invalidation becomes necessary.

1) Modified read operation

When the memory address to which the processor attempts a write is not found in the processor cache, the processor issues a memory read command and reads the data from main memory, then writes in its cache. In this case the processor's cache state is modified, so it is necessary to invalidate the cache memory of other processors. After the processor executes a memory read command, it would be possible to realize this function by executing a memory write command for cache invalidation, as explained in 2) below. However, to avoid sending unnecessary memory write commands on the bus, the M bit is attached to the memory access command.

When the processor must access memory due to a cache write miss, the M bit is set to 1 before reading from memory.

When another processor snoops the bus and performs a memory read bus operation, if the M bit is set to 1 it performs address monitoring. If a cache hit occurs in this case, that cache entry is invalidated.

2) Cache invalidation in Shared & Unmodified state

When a block registered in the copyback cache of a function unit is in Shared & Unmodified (SU) state, that block is written over, and the following command method is used for invalidating the cache of other processors.

A processor wishing to write over the block in SU state sets the M bit to 1 in the memory write command, and sends only the command and address information on the bus.

The other processors recognize the memory write command and invalidate their cache. The sole purpose of this bus operation is to invalidate cache memory, so no memory write is performed.

The sequence for cache invalidation is shown in Figure 17. In the case of a cache invalidation command, there is no particular bus slave. If, however, parity error occurs during the order command or address transfer, the bus handler detects the error. Also, if a retry operation as described in 4.7.2 occurs for the cache invalidation command, the cache invalidation operation is stopped. After the bus is obtained again, the part updated by the processor is written to shared memory.

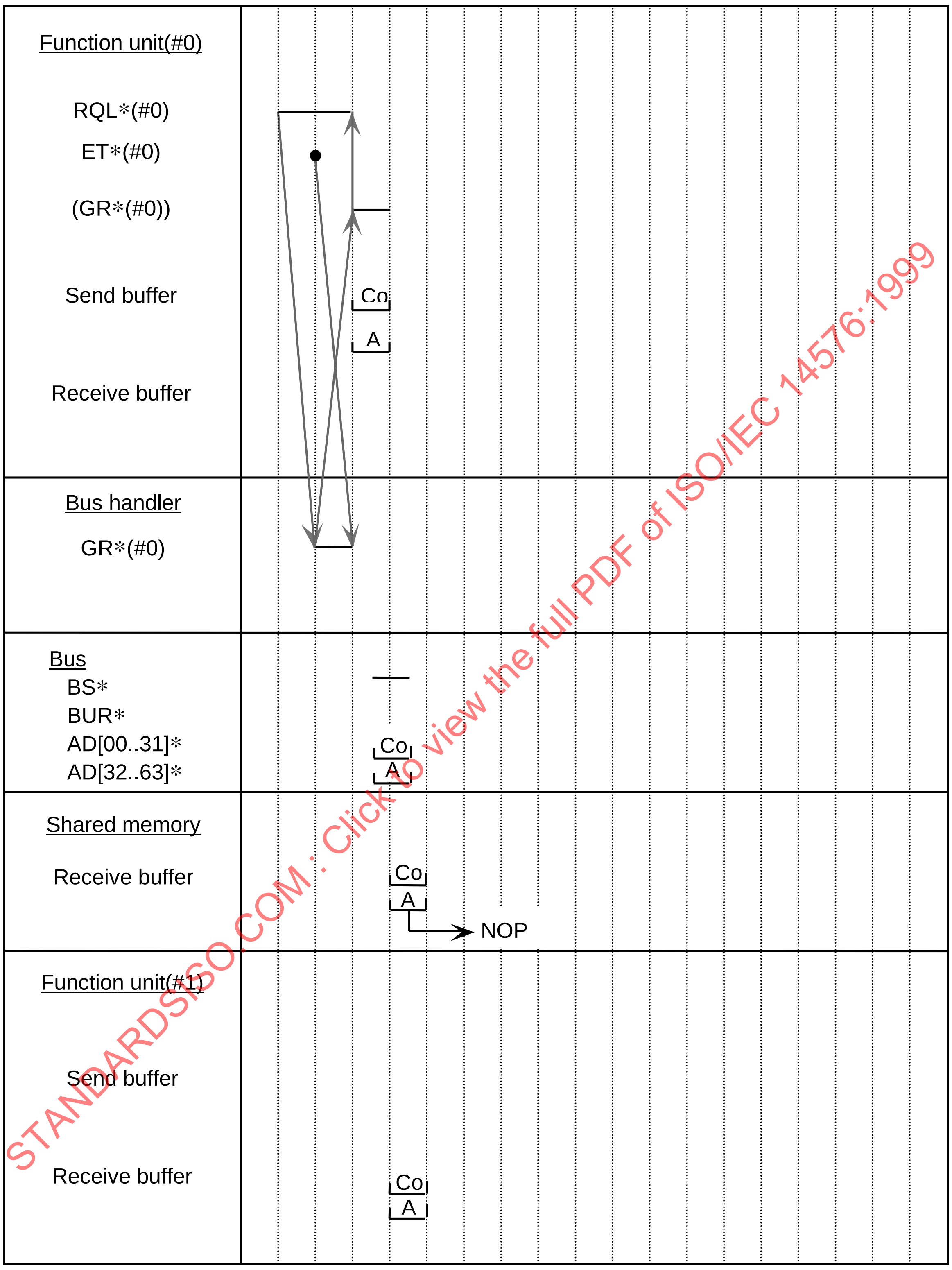


Figure 17 - Cache invalidation

4.7.2 Retry indication

In a copyback cache method, cache data are updated only in the local cache of a processor and not in a shared memory to alleviate the load on the system bus. If a local cache hit occurs when that block is read by another processor or I/O adapter, it is necessary for the read data to be sent from the local cache.

The sequence for a retry indication is shown in Figure 18.

Suppose function unit (#0), detecting a cache miss in the local cache, issued a memory block read command, but the access address was found in the cache of function unit (#1) and the state of that cache data is Exclusive & Modified (EM); in other words, the most recent block data is registered only in the cache of function unit (#1). In this case function unit (#1) asserts a RTY∗ signal and performs the writeback of that cache data.

The retry signal RTY∗ is asserted two cycles after the first cycle (command transfer cycle). In the example in this figure, a retry indication is made in response to a read order.

Control is similar even in the case of write-through operation.

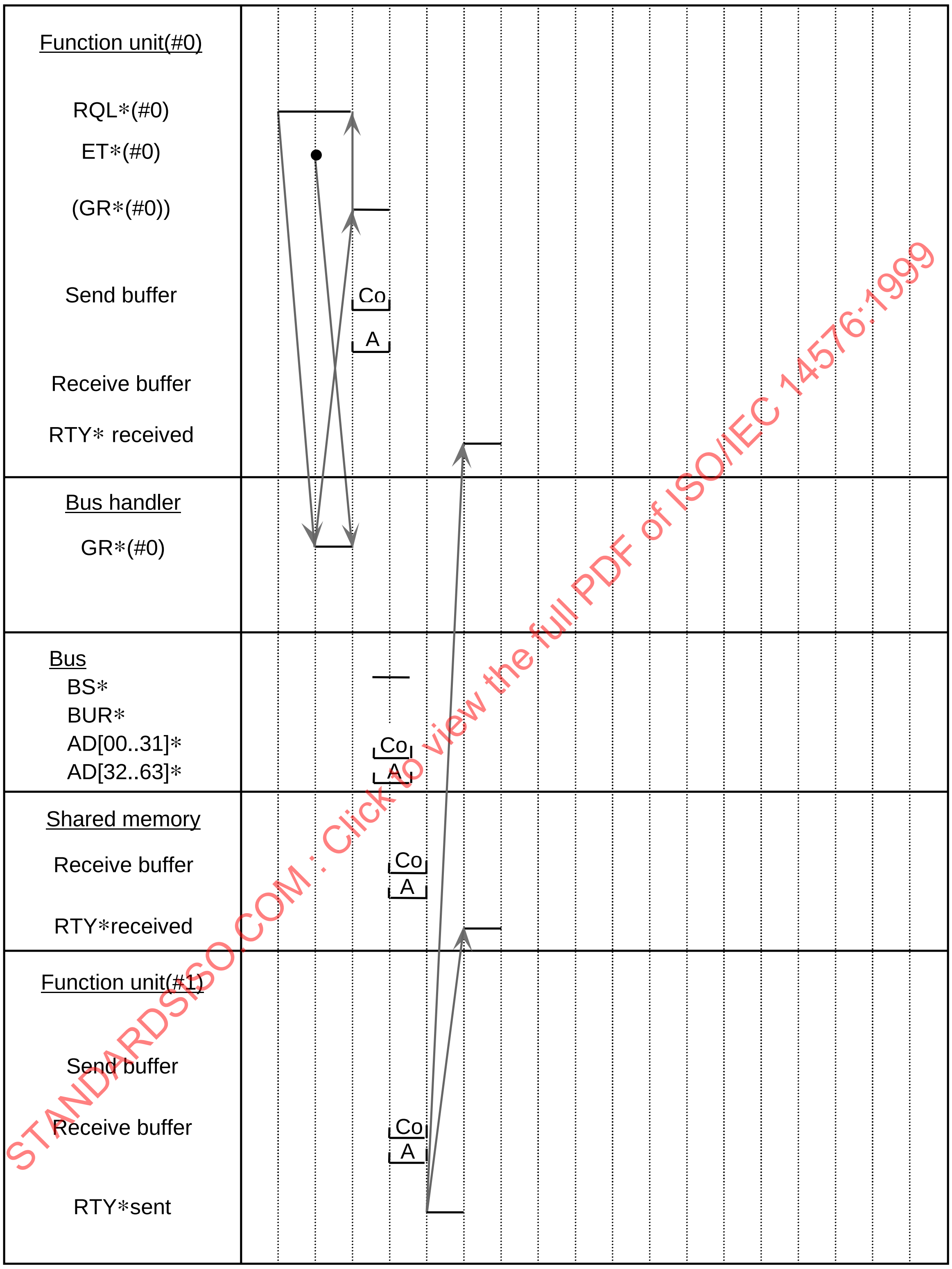


Figure 18 - Retry indication

4.7.3 Copyback and steal operations after retry indication

When the function unit corresponding to SUT receives a retry signal, it performs a retry after waiting for a predetermined holding interval to allow for copyback. Prior to the retry operation a copyback operation is performed by the function unit that asserted the retry signal.

The sequence for a copyback operation after a retry indication is shown in Figure 19. In this figure function unit (#1), which asserted the retry signal, performs a copyback of one block (32 bytes) of cache data in Exclusive & Modified (EM) state to shared memory.

The figure also shows the timing for a steal operation by function unit (#0). This function is optional, and can be used only by a system provided with a STI∗ signal.

In a system that does not support the steal inhibit signal, function unit (#0) retries access to shared memory after the copyback operation.

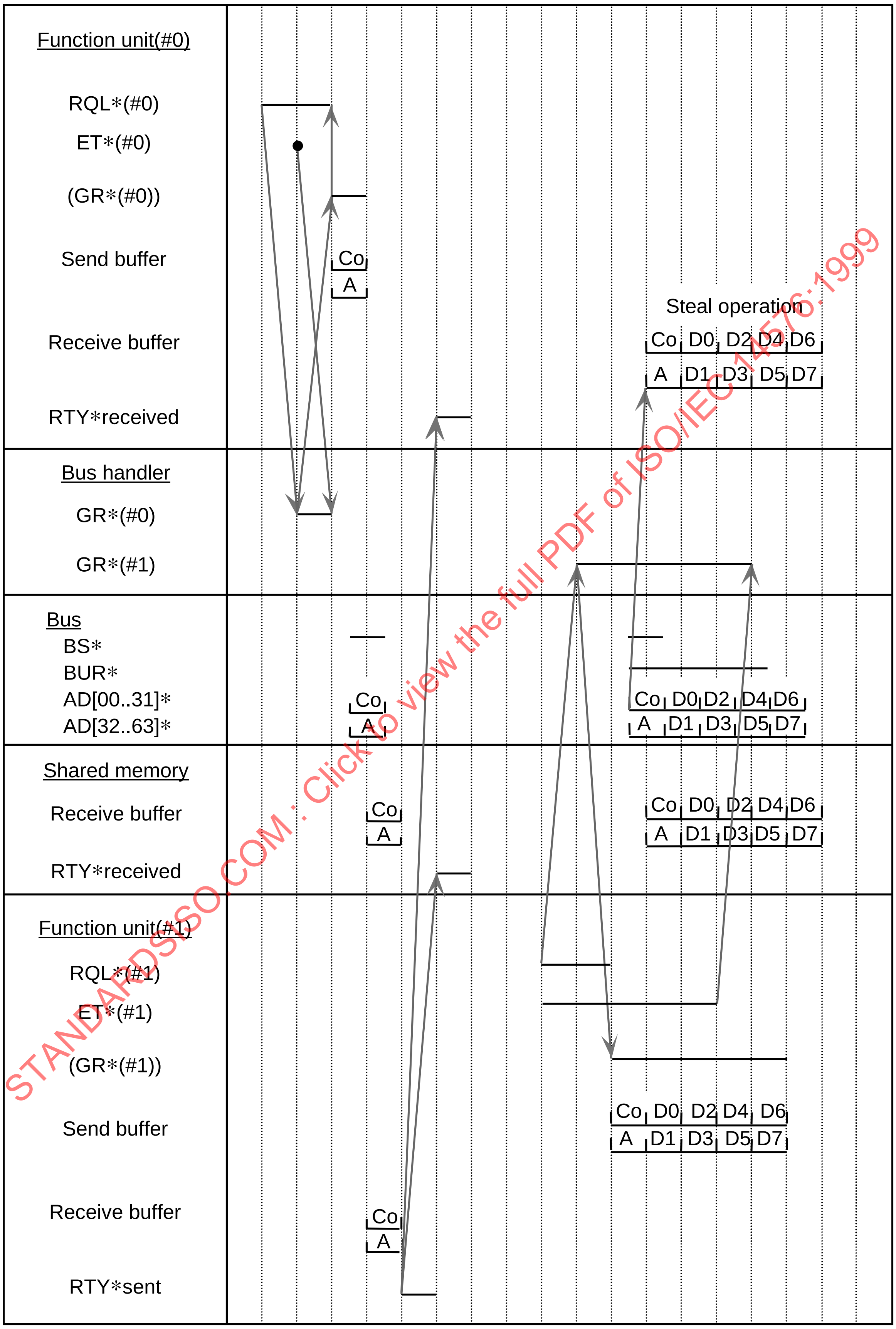


Figure 19 - Copyback and steal operations after retry indication

4.7.4 Steal inhibit operation

As explained in Chapter 5, steal operations may be inhibited for a certain interval for the sake of maintaining cache coherency. The bus operation sequence for steal inhibit indication is shown in Figure 20.

Function unit (#0) performs a read access. Next, function unit (#1), which holds the cache data in EM state at the address accessed by function unit (#0), asserts a RTY∗ signal. At this time the STI∗ signal is not asserted.

Suppose function unit (#2) performs a modified read of the same block before copyback is performed by function unit (#1). Function unit (#1) asserts a retry signal, and with the same timing asserts a RTY∗ signal. If function unit (#1) performs a copyback operation after that, function unit (#0) is able to perform a steal but steal is inhibited for function unit (#0).

If both units were permitted to perform a steal, a cache data discrepancy would result between the two copies of the same block; so in the above sequence a STI∗ signal is asserted at the time of access by function unit (#2), indicating that a steal is not allowed.

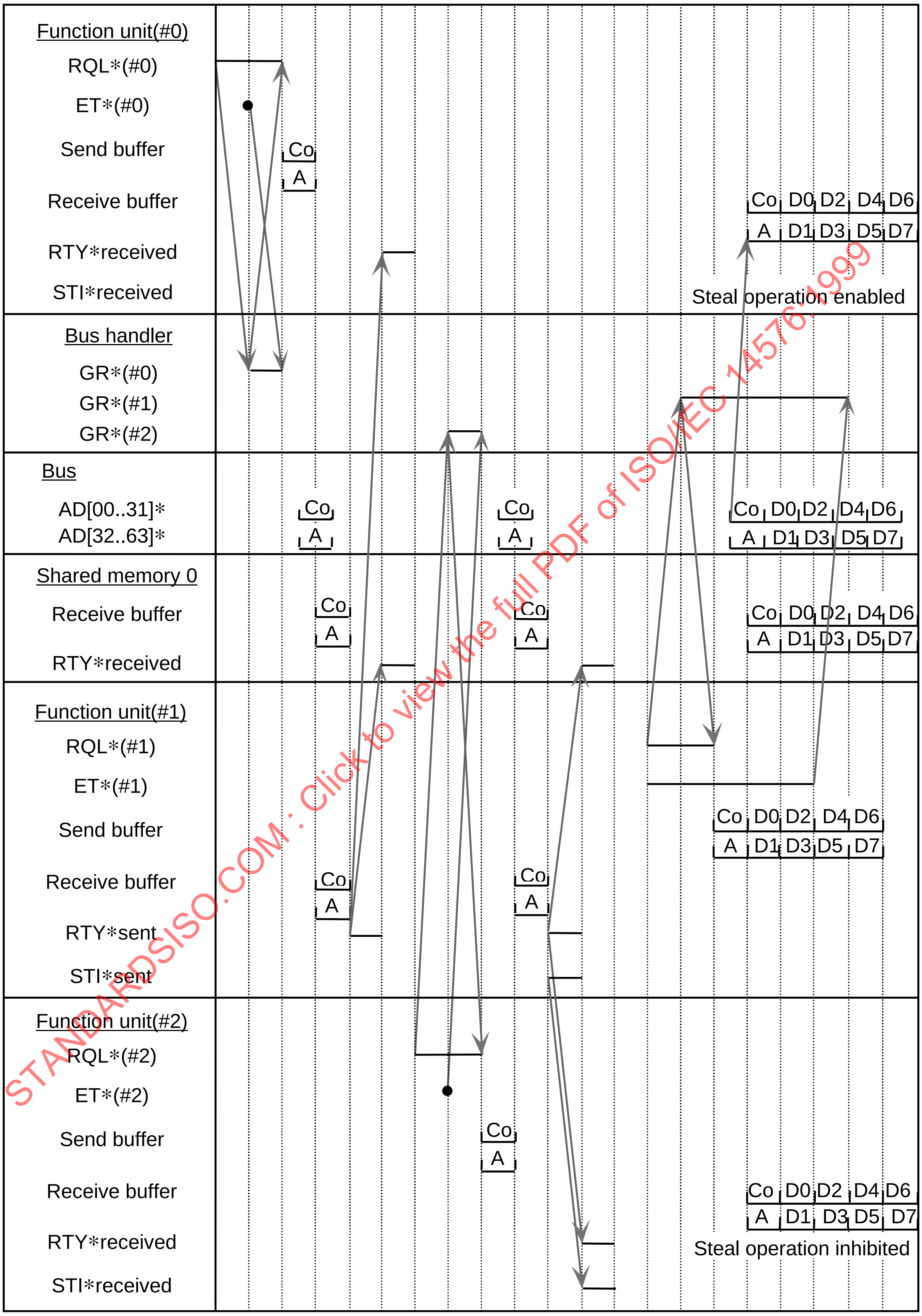


Figure 20 - Steal inhibit operation

4.8 Error Handling

This section specifies the operation when an error is reported in an answer transaction.

4.8.1 Handling errors notified in answer

An error report in an answer is issued by DUT.

When error is detected in an order transaction in a bus operation, DUT notifies SUT of the error occurrence by means of the answer transaction.

SUT receives the report and performs error processing. If necessary, SUT may perform the order transaction again.

The errors detected by DUT are as follows (See Table 7).

- Hardware error
- Illegal command
- Bus sequence error

The error report operation sequence when DUT detects error is shown in Figure 21.

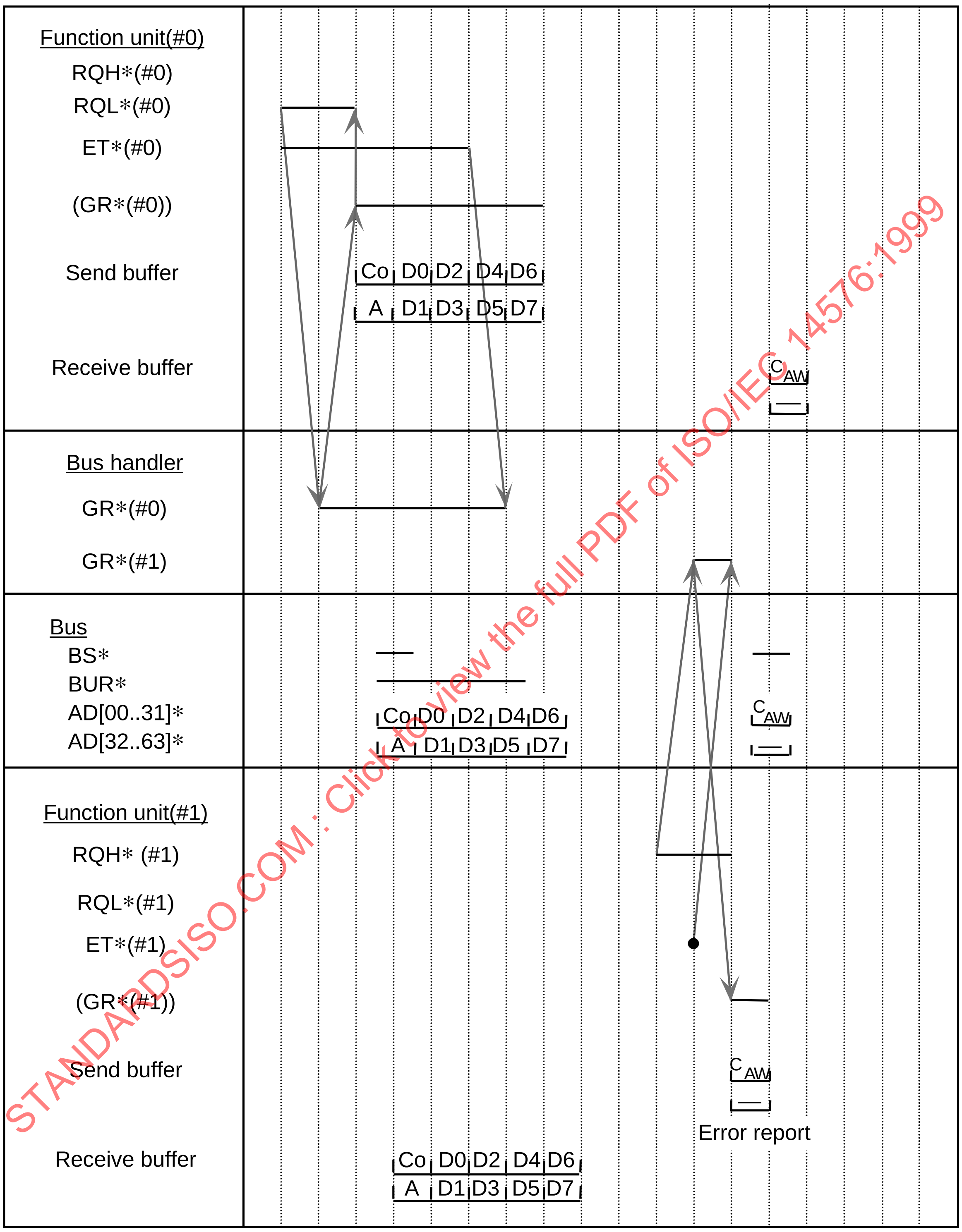


Figure 21 - Error report in answer transaction when DUT detects error

4.8.2 Other error detection

1) GRTOT (Grant Time Out)

Error is detected when SUT is unable to obtain the bus even after the prescribed waiting time has expired.

The sequence is shown in Figure 22.

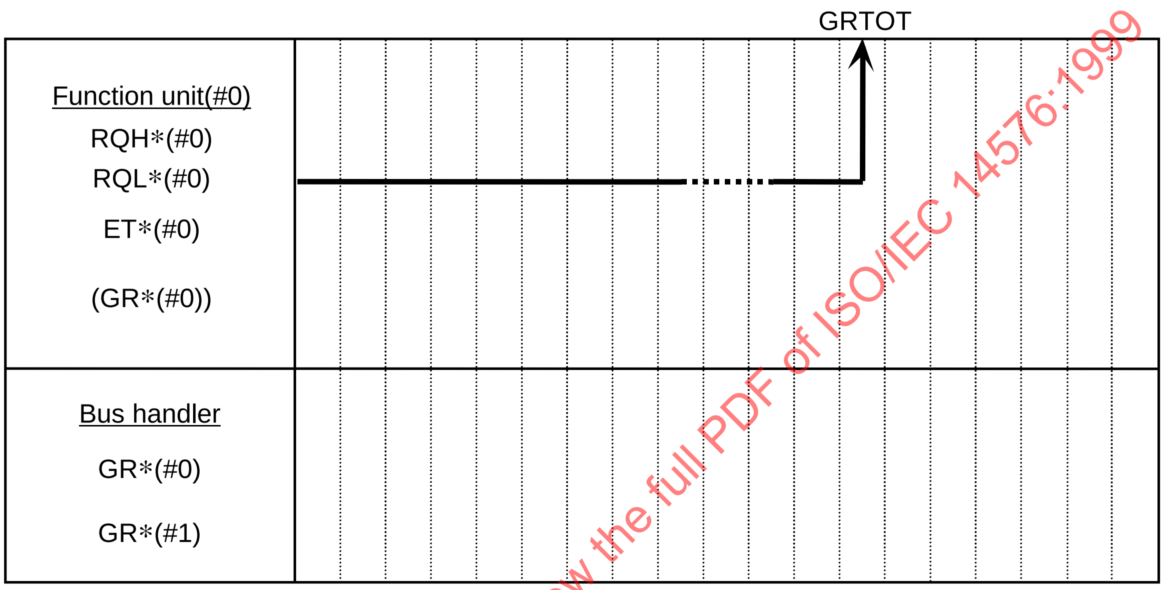


Figure 22 - When function unit (#0) detects time out

2) ASTOT (Answer Time Out)

Error is detected when SUT is unable to receive answer information even after the prescribed waiting time has expired.

The errors in 1) and 2) above are detected by a bus interface control part (BIC) in each function unit. BIC may record the type of error (GRTOT or ASTOT) and aborts the bus operation.

The length of waiting time is left as a user-dependent matter.

5. Cache Coherency Control

This chapter explains the cache coherency operations in STbus, for maintaining consistency between the contents of a shared memory and those of a cache memory in each function unit when STbus is used as a memory bus in a TCMP (tightly coupled microprocessor) system, etc.

Cache coherency is maintained in block data with a block size of 32 bytes.

As cache coherency support functions are optional, users may not necessarily implement the functions.

5.1 Cache Control Methods

The following two cache control methods are supported in STbus.

1) Write-through (consisting of the two states Invalid (I) and Shared & Unmodified (SU))

   Write data from the processor or instruction execution part is reflected directly in memory.

2) Copyback (consisting of the three states Invalid (I), Shared & Unmodified (SU), and Exclusive & Modified (EM))

   Write data from the processor or instruction execution part is not reflected directly in memory, but is updated only in the cache. Cache memory with three internal states is supported.

STbus also supports systems that use a combination of both methods. For this reason, the specifications for write-through operation also include the minimum necessary specifications for mixed use of a copyback cache scheme.

5.2 Cache Block Attributes

Each function unit can have its own cache. Each cache consists of a number of blocks, and each block is managed by the following attributes.

I:	The data registered in the block is invalid.
SU:	The data registered in the block matches shared memory. It is possible for two or more caches to have the same block.
EM:	The data registered in the block is the most recent data, and differs from the contents of shared memory. Only one block may exist in this state, and it is not shared by other caches.

To support the configuration of a copyback cache scheme using a split transfer bus, the following transient access states are defined in addition to the above cache block attributes.

I_{SU}:	Transient state during transition between I and SU, from the issuing of a read order until an answer is received.
I_{EM}:	Transient state during transition between I and EM, from the issuing of a modified read order until an answer is received.

EM_{SU}: Transient state during transition between EM and SU, from the issuing of a retry indication by another processor until copyback of the data is performed. This state occurs when a read access by another processor takes place and the most recent data in EM state is held in the cache of the local processor.

EM_I: Transient state during transition between EM and I, from the issuing of a retry indication by another processor until copyback of the data is performed. This state occurs when a write access by another processor takes place and the most recent data in EM state is held in the cache of the local processor.

5.3 Operations on System Bus

This section explains the relation between cache operations and the commands on the system bus.

The system bus commands are shown in the table below. They are realized by different bit combinations in the R/W and M fields in the memory access commands.

Table 8 - System Bus Command Types

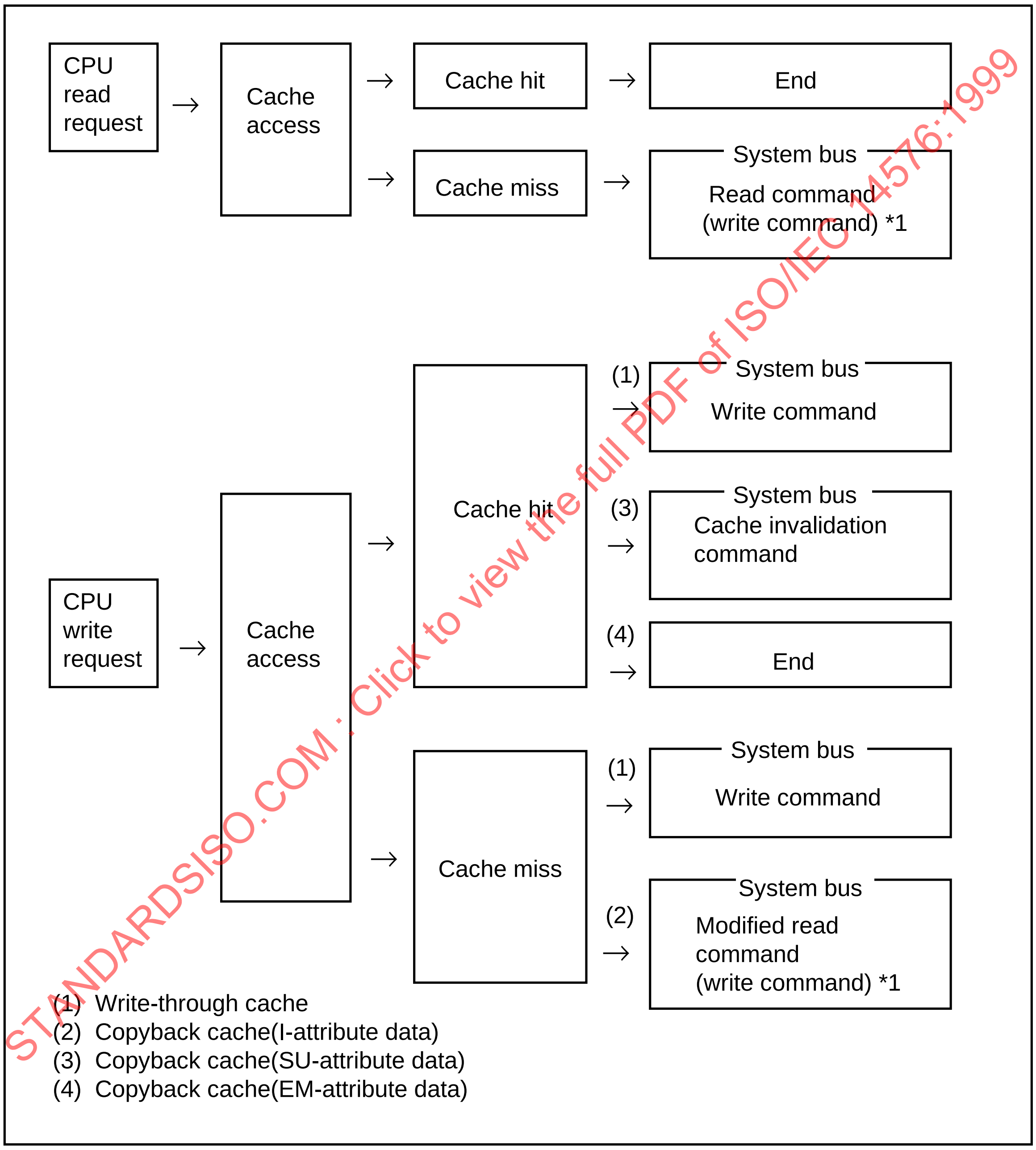
R/W	M	Bus operation
0	0	Write command
	1	Cache invalidation command (CI)
1	0	Read command
	1	Modified read command

NOTE See Figure 17 on the sequence for cache invalidation.

1) Issuing system bus commands

The issuing of system bus commands relates to CPU operation. This relation between CPU operation and each of the system bus commands is illustrated in Figure 23.

Note that system bus commands are affected by the cache control method or by the block attributes of cache data in the access area.



*1: Replace operation: An operation for registering new data in cache memory. In the case of EM-attribute data in a copyback cache, copyback must be performed before new data registration.

Figure 23 - Relation between CPU operation and commands on system bus

2) Receiving system bus commands

In order to maintain cache coherency, each unit monitors the system bus commands issued by other units. This command monitoring is called a bus snoop operation.

With a write-through cache the following operations occur.

- Write command by another unit	➩ That block is invalidated
- Cache invalidation command by another unit	➩ That block is invalidated
- Read command by another unit	➩ No action
- Modified read command by another unit	➩ That block is invalidated

In the case of a copyback cache, if a unit has EM-attribute data corresponding to a cache area to which access is made by another unit, and an answer is pending in response to an order issued for just that area, it is necessary to perform retry indication and, if supported, steal inhibition.

Details of these operations are given below in 5.4 and 5.5.

5.4 Retry Indication

A retry indication is an operation for temporarily suspending access to an EM-attribute block, when that block is being accessed by an external device. A RTY∗ signal is provided for retry indication.

A RTY∗ signal is asserted under any of the following four conditions.

1) When another function unit accesses cache data in EM state.

2) When another function unit accesses cache data in a transient state. (This does not include I_{SU} state when another unit reads the same data.)

3) When other function units receiving a cache invalidation command detect buffer full or parity error.

4) When other function units performing snoop operation detect buffer full or parity error.

The bus sequence for a retry indication is shown in Figure 18.

A unit receiving a retry indication performs the retry after waiting for a fixed interval so that access can take place after copyback is complete. The length of the wait interval and the number of retries are user-dependent matters.

5.5 Steal Operation

A steal operation is possible only in a system that supports the STI* (steal inhibit) signal. The explanation that follows does not apply to systems that do not support this function.

A function unit must satisfy the following conditions to be able to perform a steal operation.

1) In accessing the same area prior to the steal, the unit must have received a retry indication two cycles after issuing an order.

2) In the cycle in which the retry indication was received, a STI* signal cannot have been asserted.

A steal operation occurs when copyback to shared memory is performed by a function unit with an EM-attribute block, and a function unit that meets the above conditions receives the copyback data on the system bus at the same time as shared memory. It is a way of reducing unnecessary bus access.

The bus sequence for copyback and steal operations after a retry indication is given in Figure 18.

A STI* signal is asserted under the following conditions.

1) When one function unit (x) makes a retry indication in response to a memory read command by another function unit (y), and a third function unit (z) issues a modified read command for a block for which function unit x has not yet completed its copyback operation.

2) When a retry indication was made in response to a memory write command, and a third function unit accesses a block for which copyback is not complete.

3) When a block for which an answer is pending to a modified read command is accessed by another function unit.

4) When a block for which an answer is pending to a memory read command is the object of a modified read command issued by another function unit.

NOTE A retry indication is never caused by the assertion of a RTY$*$ signal (condition 1) alone, so a steal operation is not necessarily possible when a retry indication is received. When a function unit cannot perform a steal, it must itself perform a retry operation after waiting for a fixed interval.

5.6 Cache Data Management and State Transition

5.6.1 Write-through cache

A cache in a function unit must be equipped with a bus snoop function in order to maintain coherency with shared memory. That is, all function units that share the same memory must be able to monitor command and address data on the system bus, and must invalidate affected entries whenever data that is updated in shared memory corresponds to a block in cache.

The same applies to cache invalidation and modified read commands.

When write-through and copyback schemes are both used in the same system, retry operation must be supported to ensure that read and write operations will take place in the most recent data block. The reason is that when a RTY∗ signal is asserted in response to a memory access, a function unit with a write-through cache must be able to retry the read/write processing on the bus.

Write-through cache data is managed using the following two internal cache states and one transient state.

1) I (Invalid)

   The latest data has not yet been registered in cache memory. In this state there is no reaction to commands or addresses on the system bus.

   In response to a write request from a CPU, the designated block is not read. Write data is written to the external shared memory only.

   When system reset takes place, all blocks must be in this state.

2) SU (Shared & Unmodified)

   Valid data is registered for the entry. Since it is a write-through cache, the contents always match those of the external shared memory.

   In response to a write request from a CPU, the cache is written over and also shared memory is written over via the system bus.

3) I_{SU} (I → SU)

   Block receipt is pending due to a read miss. The block attribute remains I (invalid). It is necessary to monitor the system bus to determine whether there is any access to the same block. Retry indication is made in response to any access by other function units other than a read access. A steal inhibit indication (optional) is made in response to a block read in the case of a write miss by another function unit.