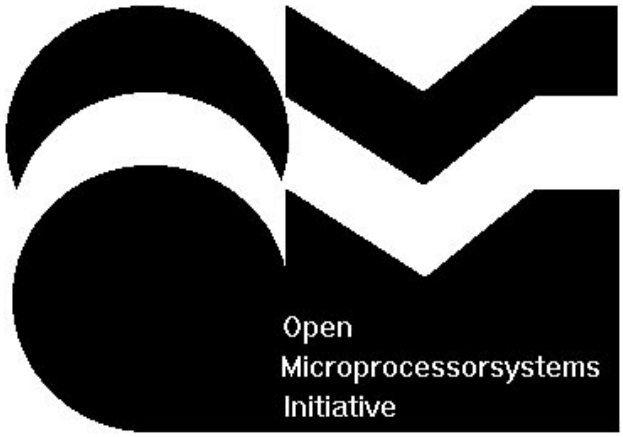


Open Microprocessor systems Initiative

DRAFT STANDARD

OMI 324: PI-Bus

Rev. 0.3d

(This forewod is not a part of the PI-Bus specification)

Foreword

The specification work for PI-Bus was carried out in the OMI/STANDARDS project (Standards Methodology and Harmonisation, 7267). OMI stands for the Open Microprocessor systems Initiative which is a part of the ESPRIT program of the European Community.

PI-Bus is one contribution to the OMI macrocell library ELI (Eurocell Library and Interfaces). ELI will support the use of macrocells of different vendors within one chip. To achieve the goal of interworking chip components it is necessary to establish standards within ELI. The OMI/STANDARDS project coordinates and organizes the contributions to ELI standards.

Partners who contributed to the PI-Bus specification are:

- Advanced RISC Machines Ltd.
- INMOS Ltd.
- Matra MHS
- Nederlandse Philips Bedrijven BV
- Siemens AG

At the time this specification was completed the working group on PI-Bus had the following membership:

Cesar Douady
David Flynn
Peter Klapproth
Graham Matthew
Thomas Niedermeier, Chair
Richard Schmid
Frits Zandveld

Other individuals who have contributed to the development are:

Gerhard Graßl
Franz Schönberger
Hans Sulzer
Peter Rohm
Thomas Steinecke
Bob Warren

Contents

CLAUSE PAGE

1. Introduction 4

1.1 PI-Bus Overview 4
1.2 PI-Bus Features 5
1.3 Scope of PI-Bus Specification 5
1.4 Conventions and Definitions 5

2. PI-Bus Fundamentals 6

2.1 Definitions of Protocol Terms 6
2.2 Pipelining (overlapping) of Bus Transfers and Bus Operations 7
2.3 Alignment of Bytes and Halfwords on PI-Bus Data Lines 8
2.4 PI-Bus Components 8
2.4.1 Master Interface (Master) 8
2.4.2 Slave Interface (Slave) 11
2.4.3 Bus Controller (Bus Control) 13
2.5 PI-Bus Signals 15
2.6 Description of PI-Bus Signals 17
2.6.1 System Signals 17
2.6.2 Bus Ownership Control Signals 17
2.6.3 Bus Operation Signals 18
2.6.4 Error Control Signals 22
2.7 Recommendations for a PI-Bus implementation 23

3. PI-Bus Operations 24

3.1 Introduction 24
3.2 PI-Bus Transfers 24
3.2.1 Single Data Read/Write Transfers 24
3.2.2 Block Transfers 26
3.2.3 Transfers of Undefined Block Length 26
3.2.4 Transfers of Defined Block Length 29

4. Transfer Termination 31

4.1 Bus Error Termination 31
4.2 Retract Termination 31
4.3 Split Termination 31
4.4 Timeout Termination 32

5. Default Grant 34

1. Introduction

The Peripheral Interconnect Bus (PI-Bus) specifies a bus protocol to be used on-chip. PI-Bus connects on-chip function blocks (modules), in particular macrocells. PI-Bus offers a wide range of functions, with primary focus on the communication requirements of various types of on-chip peripherals.

1.1 PI-Bus Overview

PI-Bus is an on-chip bus to be used in modular, highly integrated microprocessors and microcontrollers (*systems-on-chips*). PI-Bus is designed for memory mapped data transfers between its bus agents. Bus agents are on-chip function blocks (modules), equipped with a PI-Bus interface and connected via PI-Bus signals. A PI-Bus agent acts as a PI-Bus master when it initiates data read or data write operations once bus ownership has been granted to the agent. A PI-Bus agent which is addressed at a PI-Bus operation acts as a PI-Bus slave when it performs the requested data read or write operation. Typical masters are processor modules, coprocessors, or DMA controllers. Typical slaves are on-chip memory and interfaces to the external world (figure 1).

Some PI-Bus interfaces have to provide master as well as a slave functionality. E.g. a coprocessor may need to be initialised by the processor before it can be started. To initialise, the processor has to write values into the coprocessor's registers via the PI-Bus interface. In this case the PI-Bus interface operates as a slave. When the coprocessor is allowed to run, it reads from or writes to memory or communication interfaces by its own. The PI-Bus interface now acts as a master.

To operate, PI-Bus requires an additional bus controller which performs arbitration, address decoding, and timeout control functions. The bus controller may also be equipped with implementation dependent functionality, like slave access control features.

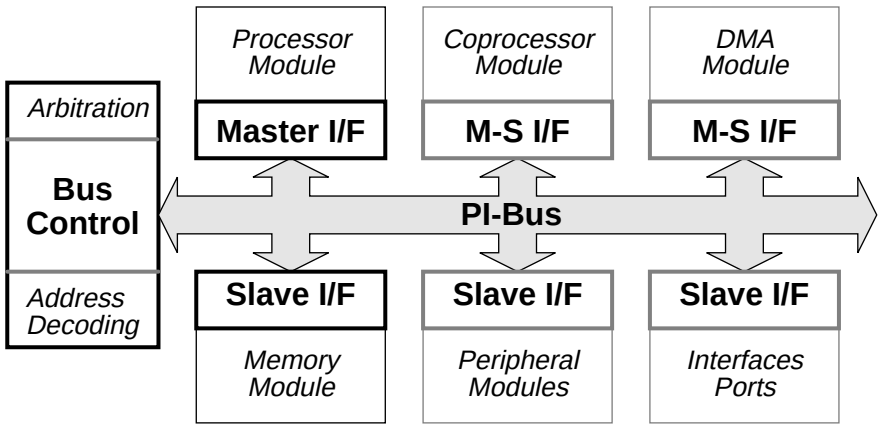


Figure 1—Modules of a single-chip system connected via PI-Bus
(I/F—means Interface, M-S—means Master-Slave)

The PI-Bus protocol is oriented towards fast PI-Bus agent accesses as well as to a high transfer bandwidth. A low-overhead protocol guarantees short response times at PI-Bus accesses

which are needed for time-critical applications. Multiple data transfers allow PI-Bus to operate at a high bandwidth.

Macrocells with a PI-Bus interface can easily be integrated into a chip layout even if they are designed by different manufacturers. The potential bus agents require only a PI-Bus interface of low complexity. Since there is no concrete implementation specified, PI-Bus can be adapted to the individual requirements of the target chip design. E.g. the widths of the address and data bus may be varied.

1.2 PI-Bus Features

The PI-Bus is designed with requirements of high-performance systems in mind. Main features of the PI-Bus are:

- Processor independent
- Demultiplexed operation
- Clock synchronous
- Peak transfer rate of 200 Mbytes/s (@ 50 MHz bus clock)
- Address and data bus scalable (up to 32 bits)
- 8-/16-/32-bit data accesses
- Broad range of transfer types from single to multiple data transfers
- Multimaster capability

PI-Bus does **not** provide:

- Cache coherency support
- Broadcasts
- Dynamic bus sizing
- Unaligned data accesses

1.3 Scope of PI-Bus Specification

The PI-Bus specification covers the PI-Bus protocol. It describes which set of transfers and signals at least have to be supported by PI-Bus agents. It also lists options which may be supported by a PI-Bus agent.

The PI-Bus specification does not regulate implementation issues. In particular, it does not specify the layout of a PI-Bus interface or define electrical parameters.

1.4 Conventions and Definitions

master	stands for the master functionality of a module's PI-Bus interface.
slave	stands for the slave functionality of a module's PI-Bus interface.
bus control	stands for all control components which are required by PI-Bus but are not part of a module's PI-Bus interface.
CLK ⌐↑	is a symbol for the rising edge of the bus clock **CLK**.

PI-Bus signal names are always written in **bold** letters, like **READ**.

2. PI-Bus Fundamentals

2.1 Definitions of Protocol Terms

transaction

The term transaction is intended to be used in a PI-Bus system with the possibility of split mode transfers. If the split mode is implemented a transfer may be broken up by a slave into two normal PI-Bus transfers (transaction). Then, the first transfer only transmits a read or write request from the requesting agent to a target agent, the responder. In case of a write request data are also transmitted. The responder will send in an own bus transfer an acknowledge information and in the case of a read the requested data.

bus transfer

A bus transfer is an exchange of one or multiple data (byte, halfword, word) between a master and normally one slave.

A bus transfer begins when a master requests bus ownership from bus control. The master starts the data transfer with a bus operation once bus ownership has been granted by bus control. A master may chain multiple bus operations within a bus transfer. The number of chained bus operations may range from one to indefinite. Chained bus operations may also be referenced as *block transfer.*

During normal operation a bus transfer is terminated under slave control with the last data cycle of the last bus operation.

In the event of errors a bus transfer may be aborted by the selected slave or by bus control during any data cycle.

Bus control may grant bus ownership to another master only during or after transfer termination or after transfer abort.

bus operation

Within a bus operation exactly one data (byte, halfword, word) is exchanged between a master and a slave. A bus operation is either a read from or a write to a slave.

A bus operation consists of at least two bus cycles. It is started by a master during an address cycle and it is terminated/aborted by the selected slave or by bus control during a data cycle.

To chain bus operations a master may indicate in the address cycle that the current bus operation is followed by another one. Otherwise the current bus operation is the last in the bus transfer. The PI-Bus protocol requires to pipeline (overlap) chained bus operations.

bus cycle

A bus cycle is the smallest unit of the PI-Bus protocol and corresponds to one bus clock period. It starts with the rising edge of the bus clock.

There are three of bus cycle types: *address*, *data* and *idle cycle*.

Address and data cycles of consecutive bus operations in a block transfer do overlap.

address cycle During an address cycle the master supplies the address, opcode, lock and read information for the current bus operation.

At chained bus operations a master issues the address cycle of next bus operation once the data cycle of the current operation has been started. A master shall repeat the address cycle of next bus operation as long as the slave indicates by a wait acknowledge code that it had been busy and couldn't accept the address cycle information.

A slave is selected at the end of an address cycle. Slave selection is always inhibited

— as long as the slave issues a wait acknowledge code in the data cycle of a previous bus operation
— if the started bus operation is a no-operation
— if the issued address was not within the allowed address space.

Bus grant for the current master may be released during an address cycle. **However, a new master shall not yet be granted**!

data cycle In the data cycle, data is read from or written to a slave.

The data cycle of a bus operation is repeated if the selected slave issues a wait acknowledge code (*data wait cycle*). Bus control may grant bus ownership to a new master in the last data cycle of a bus transfer indicated by the selected slave via a ready acknowledge code.

idle cycle The bus is an idle cycle if neither an address nor a data cycle is performed. Grant may be given to a master.

2.2 Pipelining (overlapping) of Bus Transfers and Bus Operations

To achieve highest performance PI-Bus allows to pipeline (overlap) some phases of its protocol. A master may start to request bus ownership in parallel with a current bus transfer of another master. If no master requests the bus, grant may also be given to one master in advance. Once ownership has been granted, the requesting master may immediately start with the first bus operation of the bus transfer (figure 2).

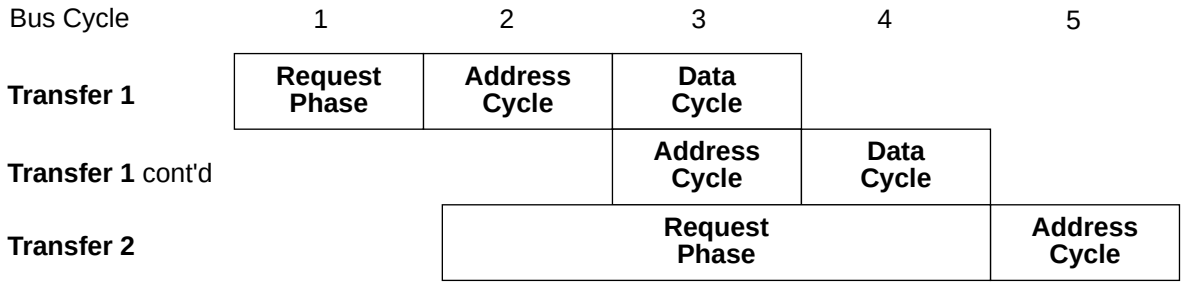


Figure 2—Pipelining (overlapping) of bus transfers and bus operations

During a block transfer the information of the address cycle of the next bus operation is issued in parallel to the data cycle(s) of the current bus operation. In case of a very fast slave, which doesn't insert data wait cycles, a peak data rate of one data per bus cycle can be achieved.

2.3 Alignment of Bytes and Halfwords on PI-Bus Data Lines

PI-Bus interfaces should implement right aligned transfers of bytes and halfwords as shown in figure 3. **D[7:0]** should be used for single bytes and the least significant byte (LSByte) of halfwords and words. **D[15:8]** should be used for the most significant byte (MSByte) of a halfword. **D[31:24]** should be used for the most significant byte of a word (see figure 3).

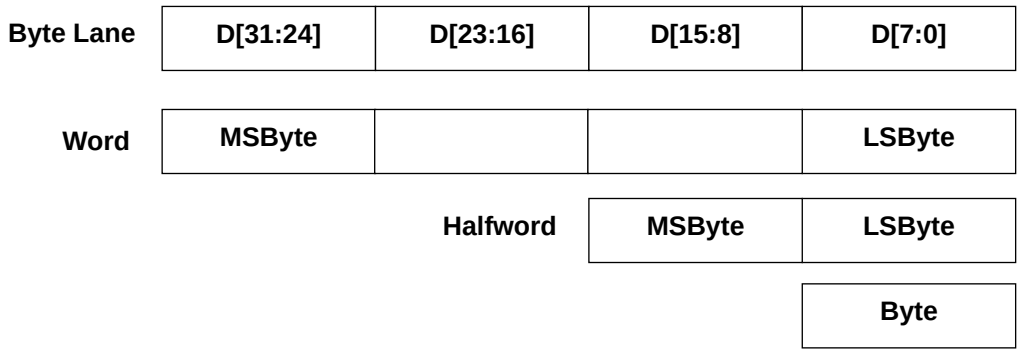


Figure 3—Right Alignment of bytes and halfwords on PI-Bus data lines.

Right alignment is the recommended byte to byte-lanes assignment at PI-Bus due to the following reasons:

- No endian format dependency. PI-Bus agents of different vendors can easily be integrated.
- Low cost integration of “smaller” agents. The interface of masters with internal buses smaller than 32 bits doesn't need to be expanded to 32 bits. It is also not necessary to route all data bus lines to all masters.
- Agent internal data buses may be connected directly to PI-Bus data bus lanes.

NOTE—If a system implements only 32-bit agents and if these agents have a common endian format then it might be preferred to transfer bytes and halfword transfers in this system on natural byte lanes. It shall be indicated in a PI-Bus agent's data sheet if that agent doesn't support right alignment!

2.4 PI-Bus Components

PI-Bus components are master interfaces (masters), slave interfaces (slaves) and a bus controller (bus control). A PI-Bus agent can be PI-Bus master, PI-Bus slave or both.

2.4.1 Master Interface (Master)

A PI-Bus agent needs an interface with master functionality if it is an active system module which performs read and write accesses to other modules. PI-Bus allows to have more than one bus master interface. In a multi-master configuration an arbitration logic shall be imple-

mented to select one of several competing master interfaces.

The diagram in figure 4 shows possible states and state transitions of a PI-Bus master interface. Additional requirements for the behaviour of a master interface are below summarised:

Reset

Any activation of the low active reset signal (**RESETN**) shall force the master interface into a reset state until the reset signal has been released. All bus drivers shall asynchronously be deactivated as long as **RESETN** is active. The reset signal is released synchronously to the rising edge of the bus clock **(CLK** ↓).

Idle

An idle master shall keep all its bus drivers deactivated. To get bus ownership a master has to set the request signal (**REQ**).

A master with default grant option, which is given a default grant, may start a bus transfer in any granted cycle It shall enter the first address cycle and at the same time assert its request line.

Request

A master is granted bus ownership when its grant line is activated by bus control at **CLK** ↓. A requesting master which is granted bus ownership shall start a bus operation (at least a no-operation) in the cycle following the assertion of the grant.

Address

A master shall drive the address, read, and opcode information (**ADR[31:2]**, **READ**, **OPC[3:0]**) in the address cycle of a bus operation. In case of a no-operation only **OPC[3:0]** may be driven. To chain bus operations for a block transfer a master shall assert the lock signa (**LOCK**).

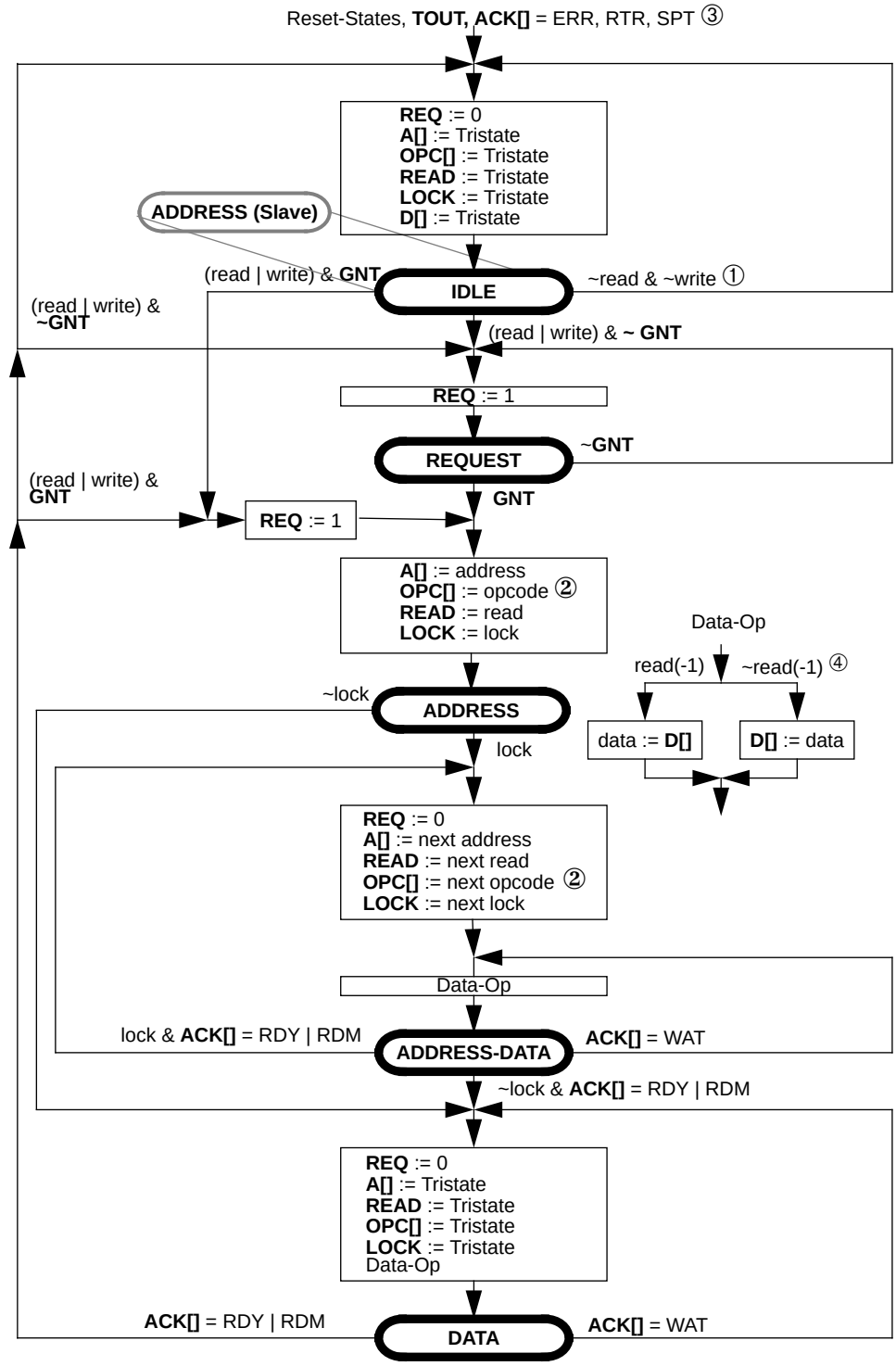
Address-Data

In the case of pipelined bus operations a master shall repeat the address cycle of the new bus operation as long as the data cycle of the previous bus operation isn't completed with a **Ready** acknowledge code (RDY, RDM) returned by the selected slave.

If the direction changes within a data transfer (read or write) a no-operation bus cycle shall be inserted to avoid contention of the data and acknowledge bus lines.

An active master which receives the slave acknowledge code **Error** (ERR) or **Retract** (RTR) shall abort the current bus cycle and release all bus lines during the next bus cycle. After having received an RTR code, a master shall wait at least one bus cycle before issuing a new request.

An active master which sees an active timeout signal (**TOUT**) shall abort the current bus cycle and release all bus lines during the next bus cycle.



① address, opcode, data, read, write and lock are module internal signals. They control the behaviour of a module's master interface

② To insert a delay opcode may become NOP.

③ After detecting **ACK[]** = ERR, RTR, SPT or an active **TOUT** a master interface shall always deactivate its drivers and enter the IDLE state.

④ The data operation depends on the status of the read signal of the bus operation currently in the data cycle.

Figure 4—State diagram of a PI-Bus master interface

2.4.2 Slave Interface (Slave)

A PI-Bus agent needs an interface with slave functionality, if it is a passive system module, addressed via reads and writes by other modules. A slave interface is the addressees of a master interface initiated transfer. Each slave interface is selected and activated via a dedicated slave select signal **(SEL**). This signal is normally decoded from the address issued by the active master.

The diagram in figure 5 shows possible states and state transitions of a PI-Bus slave interface. Additional requirements for the behaviour of a slave interface are below summarised:

Reset

Any activation of the low active reset signal (**RESETN**) shall force the slave interface into a reset state until the reset signal has been released. All bus drivers shall asynchronously be deactivated as long as **RESETN** is active. The reset signal is released synchronously to the rising edge of the bus clock **(CLK** ↓).

Address

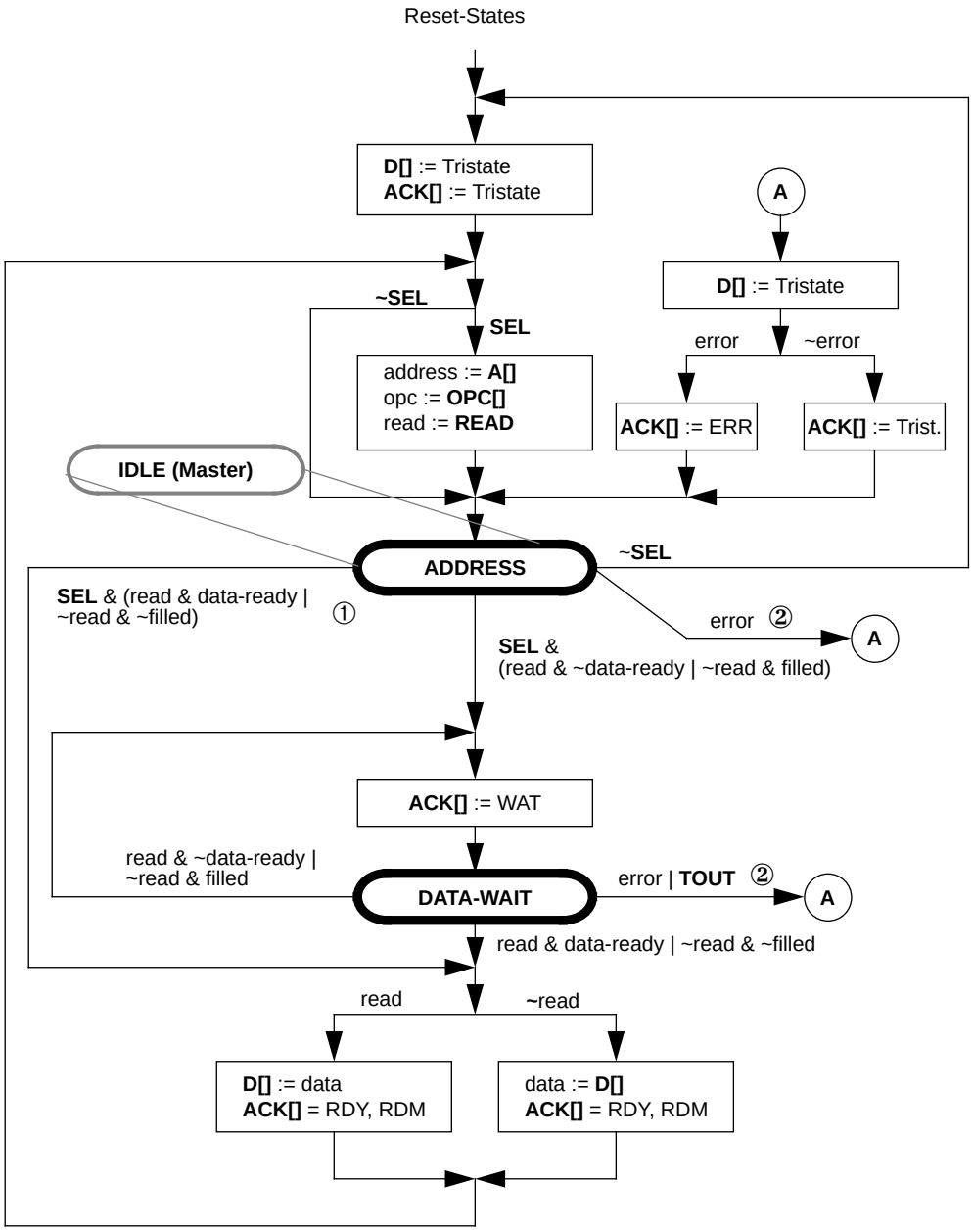
An idle, not selected slave shall keep all its bus drivers deactivated. A slave is selected when its select signal (**SEL**) is asserted.

A selected slave shall start to drive an acknowledge code (**ACK[2:0]**) during the following bus cycle(s) until the bus operation is finished.

In case of an error condition (wrong opcode, wrong address, no data at reads, buffer filled at writes) a selected slave shall drive an **Error** acknowledge code (ERR) and release all driven bus lines during the next bus cycle.

Data-Wait

A selected slave which sees an active timeout (**TIMEOUT**) signal shall release all driven bus lines during the next bus cycle.



① read, data-ready, filled, address, opc and error are module internal signals. They control the behaviour of a module's slave interface.

② After detecting an active error or **TOUT** signal a slave interface shall always abort the current activity and enter the

Figure 5—State diagram of a PI-Bus slave interface

2.4.3 Bus Controller (Bus Control)

PI-Bus needs a bus controller for operation. It is intended that all bus control functions are implemented centralised. Figure 6 shows possible states and transitions of a PI-Bus controller.

The requirements for a bus control implementation are:

arbitration

Bus control has to arbitrate which master is granted the requested bus ownership. If no master requests the bus, grant may be given by default to one master. **Only one grant may be active at any time**.

A grant may only be given if

— the bus is idle

— it is a data cycle of a no-operation (NOP) and the lock signal (**LOCK**) is not set

— it is a data cycle, the slave is returning a ready acknowledge code (RDY, RDM) and the lock signal (**LOCK**) is not set.

address decoding

To determine the target slave of a bus operation, bus control decodes in the address cycle upper bits of the address issued by the granted master. The corresponding slave select signal (**SEL**) is then set late in the address cycle. It depends on the memory map of the implementation which address bits are decoded.

The generation of the slave select signal is inhibited if

— the master starts a no-operation

— in the case of an pipelined bus operation the slave is returning a **Wait** acknowledge code (WAT) in the data cycle of the previous bus operation.

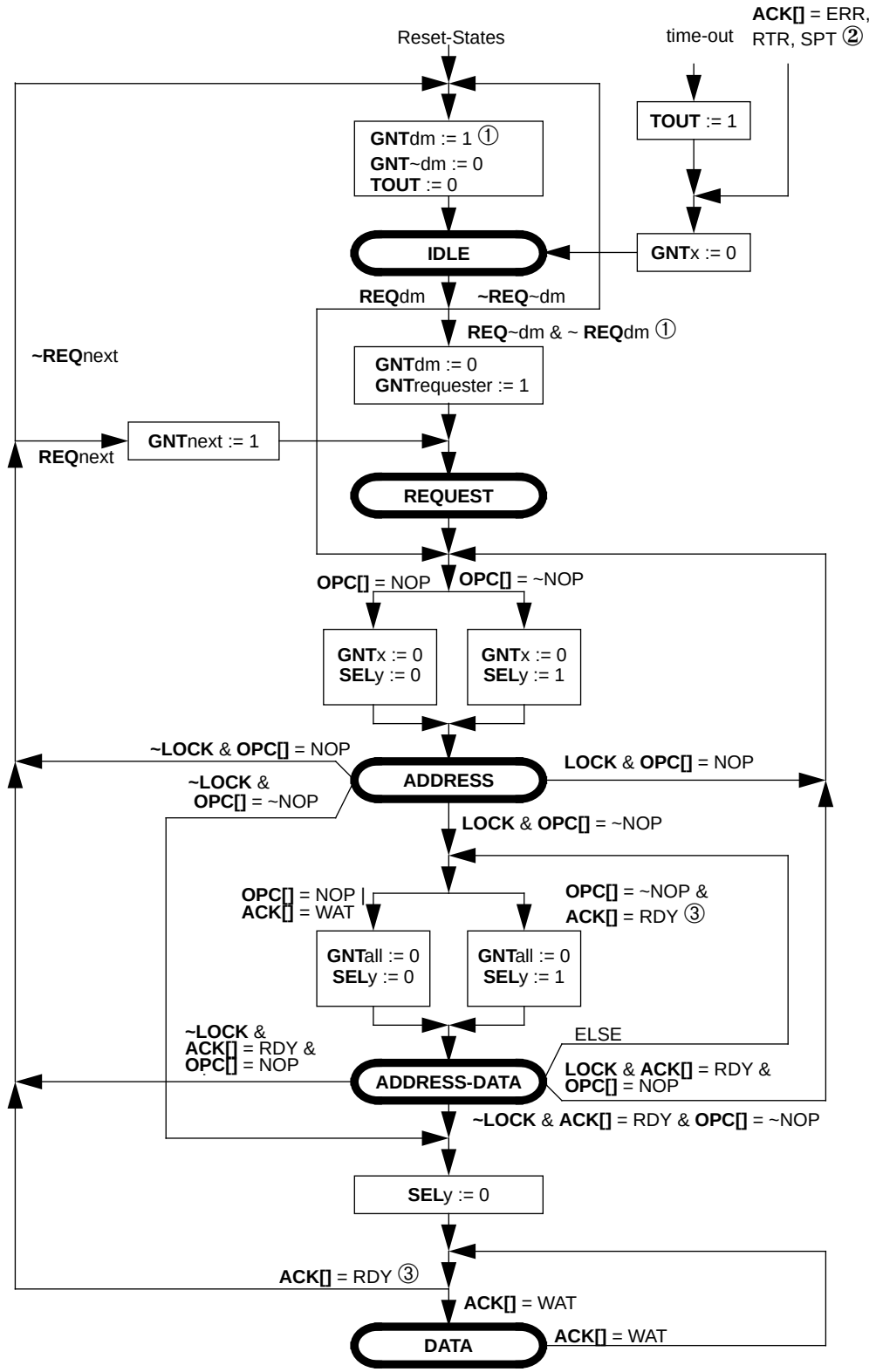
The address decoding mechanism shall also guarantee that a built-in “default slave” is selected if a master issues an address which is not mapped to a slave. The “default slave” shall generate an **Error** acknowledge code (ERR).

timeout control

The timeout mechanism is intended for bus operations which are not completed by the selected slave with a **Ready** acknowledge code (RDY, RDM). Bus control should assert the timeout signal after an implementation dependent number of data wait cycles.

slave access control

This feature may be implemented in bus control to supervise slave accesses. If a master initiates an access to a forbidden slave (e.g. restricted via the process identifier) , the “default” slave may become active with an **Error** acknowledge code (ERR) returned to the master.



① dm stands for a Default Master, x for an active master, and y for a selected slave.
② An active time-out signal (internal signal) or an **ACK[]** = ERR, RTR, SPT at the ADDRESS-DATA and DATA states shall always lead to the IDLE state.
③ RDM is always treated like RDY and left out of this diagram.

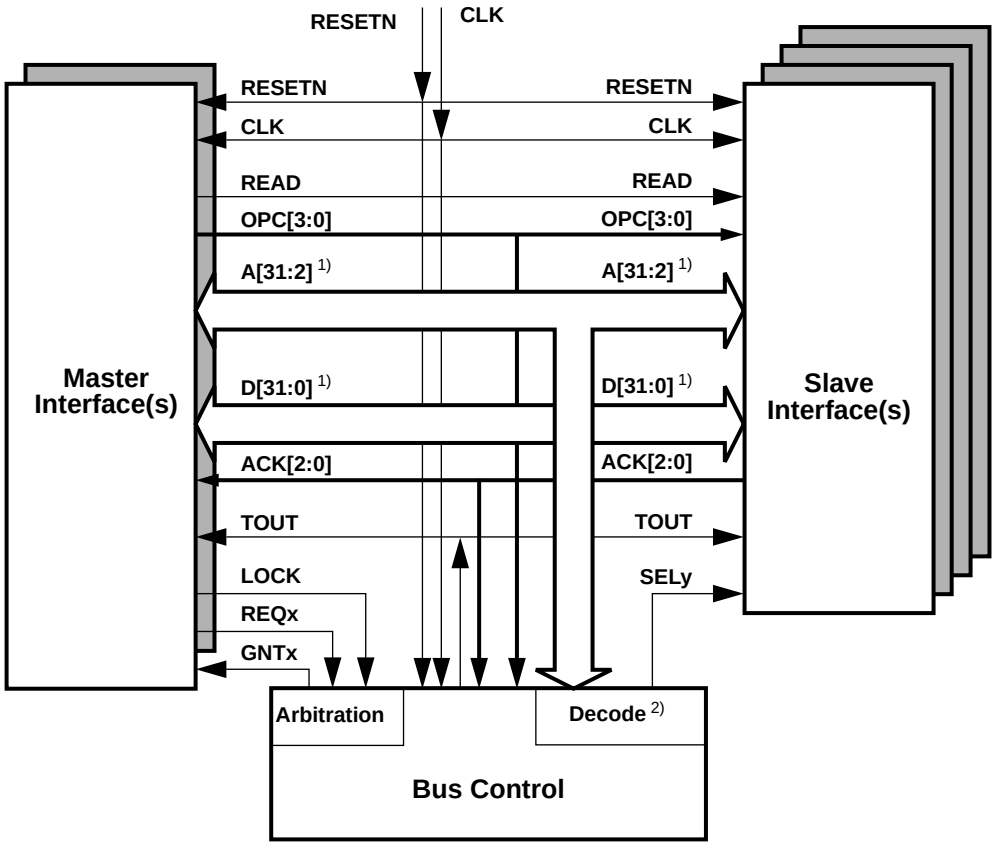
Figure 6—State diagram of a PI-Bus bus controller

2.5 PI-Bus Signals

Figure 7 shows PI-Bus signal lines routed between PI-Bus components, a master interface, a slave interface and the central bus control. **RESETN** and **CLK** are system signals connected to every bus component. **LOCK**, **READ**, **OPC[3:0]**, **A[31:2]**, **D[31:0]**, and **ACK[2:0]** are driven by more than one source (bused signal lines). For every master interface exists a **REQ-GNT** signal pair and for every slave a select signal **SEL**. The **TOUT** signal is driven by bus control and routed to every master and every slave.

The maximum width of the address bus **A[31:2]** depends on the system memory space of the implementation. The upper address bits are mapped to the slave select signals. The lower bits address slave internal memory locations, like registers, FIFOs, or RAM. A master has to be connected to all implemented address lines, a slave only to those address lines which are needed for the slave internal decoding.

The maximum width of the data bus **D[31:0]** is determined by the largest data type (byte, halfword, word) which will be transported over PI-Bus. The data types processed by an agent determine which data lines have to be connected to that agent. A byte agent drives **D[7:0]**, a halfword agent **D[15:0]** and a word agent **D[31:0]**.



NOTES

1) The widths of the address and the data bus depend on the implementation of the PI-Bus.

2) Address decoding may also take place close to the slave interface. In that case bus control does no longer control select signal assertion and the problem may arise that two slaves may compete for the bus. This has to be solved by an appropriate configuration of the PI-Bus system (e.g. insertion of no-operations).

Figure 7—Routing of PI-Bus signals between PI-Bus components

Table 1 lists attributes of every PI-Bus signal. Section 2.6 describes each signal in detail.

Table 1—Attributes of PI-Bus signals

Signal Name	Driven by	Driven when (in bus cycle)	Received by	Receiv'd when	Comments
RESETN	System		All bus components		Asynchronously activated, synchronously deactivated, ***active low!***
CLK	System		All bus components		Reference bus clock
REQx	Master x	early	Bus Control	mid	Master request
GNTx	Bus Control	mid	Master x	late	Master grant
LOCK	Master	early	Bus Control	mid	Locks consecutive bus cycles, inhibits arbitrat. for next bus cycle
READ	Master	early	Slave	mid	Read/Write
OPC[3:0]	Master	early	Slave, Bus Control	mid	Operation code
A[31:2]	Master	early	Bus Control, Slave	mid	Address bus, Width as required
D[31:0]	Master, Slave	early	Slave, Master	mid	Data bus, Optional: **D[31:8]**
ACK[2:0]	Slave, Bus Control	early	Master	mid	Slave response code
SELy	Bus Control	mid	Slave y	late	Slave select
TOUT	Bus Control	early	Master, Slave	mid	Timeout signal

REQ, **GNT**, **LOCK**, **READ**, **SEL**, and **TOUT** are active high signals. A low (“0”) level defines the inactive and the high (“1”) level defines the active signal state. **RESETN** is low active!

PI-Bus signals shall always carry defined logical values to reduce power consumption. This must be guaranteed by the implementation of the PI-Bus and may be achieved by a hold mechanism at the PI-Bus lines. **The PI-Bus protocol makes no use of held signal levels.**

NOTES
1—early = early in cycle (signal has to be driven only a few gate delays after clock rising edge)
mid = mid of cycle (reception of this signal is not very critical in time at the agent side.
late = late in cycle (reception of this signal is time critical. Only a few gate delays are allowed until registration).
2—The PI-Bus protocol has been defined such that within one bus cycle not more than one signal transfer on a bus lane (e. g. **A[31:2]**) plus one signal transfer on a point-to-point line (e. g. **SEL**y) can occur.
3—PI-Bus signals driven by different bus agents are defined in a way that no signal contention shall occur.

2.6 Description of PI-Bus Signals

2.6.1 System Signals

RESETN

Hardware Reset: During the circuitry power-up phase **RESETN** shall asynchronously, during normal operation synchronously to **CLK** be activated. **RESETN** is always **CLK** synchronous deactivated**.**

RESETN=0 Reset internal state of PI-Bus interfaces and of bus control. Busdriver outputs connected to **LOCK, READ, OPC[3:0], A[31:2], D[31:0]** and **ACK[2:0]** shall be disabled.

RESETN=1 Normal operation

NOTES
1—**RESETN** is active low.
2—Further extensions of the reset and debug/emulation behaviour of PI-Bus interfaces are likely to be defined as an extension to **RESETN** according to the results of Reset / Initialization / Debug activities within OMI/Standards.

CLK

Bus Clock: Bus cycle timing is referenced to the rising edge of the bus clock (**CLK** ↑).

2.6.2 Bus Ownership Control Signals

REQx

Master Request: **REQ** is driven by a master early in a bus cycle to request bus ownership from bus control. If due to a default grant bus ownership has already been given to the master in the previous bus cycle, the master indicates via **REQ** whether it takes bus ownership to start a bus operation.

REQ=1 Request for bus ownership /
Bus operation started if default grant is already given.

REQ=0 No request for bus ownership /
No bus operation started if default grant is already given.

A granted master has to reset **REQ** at latest during the first data cycle of the first bus operation if no further bus request shall be issued.

NOTE—The number of **REQ** lines depends on the number of masters in the system. Each master has a separate **REQ** line to bus control.

GNTx

Master Grant: Bus control asserts **GNT** after arbitration to indicate to a master that its request for bus ownership was accepted. **GNT** is asserted in idle cycles or in the last data cycle of a bus transfer. Only one **GNT** may be active at the end of a bus cycle. The granted master owns the bus in the next bus cycle and shall start a bus operation with an address cycle (at least a NOP operation). Earliest during this

address cycle bus control may release **GNT.**

GNT=0 Master does not get bus ownership in the next bus cycle.

GNT=1 Master gets bus ownership in the next bus cycle.

Optionally, bus control may implement a *default grant* mechanism. In this case a grant may be given to a master without a previous request.

NOTES
1—The number of **GNT** lines depends on the number of masters in the system. Each master has a separate **GNT** line from bus control.
2—The arbitration algorithm is intentionally left open for the implementation.

LOCK **Lock (chain) bus operation**s: A master drives this single bus line early in an address cycle to indicate to the bus control that another bus operation shall be chained to the current one. When **LOCK** was asserted, bus control shall not give any **GNT** in the following data cycle.

LOCK=0 Bus control may perform arbitration and assign bus ownership on termination of the current bus operation.

LOCK=1 Master wants to chain the next bus operation and does not give up bus ownership. Arbitration shall be inhibited.

NOTE—Typically, this signal is used by a master to execute longer data transfers which should not be interrupted. This may also be used e.g. for a read/modify/write operation. Bus transfers may be of defined or undefined length.

2.6.3 Bus Operation Signals

READ **Read Operation**: This single bus line is driven by the granted master early during an address cycle to inform the selected slave about the direction of the data transfer during the following data cycle.

READ=0 Write - Data shall be transferred from the master to the slave.

READ=1 Read - Data shall be transferred from the slave to the master.

In case of a no-operation (NOP), **READ** may remain undriven.

NOTE—**READ** should be sampled by the slave at the end of an address cycle.

OPC[3:0] **Bus Operation Code**: The opcode bus lines are driven by the granted master early in an address cycle to inform bus control and the selected slave about the characteristics of the current bus operation:
- access to special control address space
- data width
- byte and halfword selection

- block transfer word count

Table 2—Operation codes

OPC[3:0]	Identifier	Description
000X	NOP	No-operation
0010	WDU	Word transfer (undefined length block transfer)
0011	WDC	Word transfer control/special address space
01mn	WDx	Word transfer (defined length block transfer) m,n = 0,0; 2 words (x=2) m,n = 0,1; 4 words (x=4) m,n = 1,0; 8 words (x=8) m,n = 1,1; 16 words (x=16)
10a0	HWx	Halfword transfer (undef. length) a = 0; Halfword 0 (x=0) a = 1; Halfword 1 (x=1)
10X1	RSD	Reserved
11ab	BYx	Byte transfer (undefined length) a,b = 0,0; Byte 0 (x=0) a,b = 0,1; Byte 1 (x=1) a,b = 1,0; Byte 2 (x=2) a,b = 1,1; Byte 3 (x=3)

In case of a block word transfer the corresponding opcode shall be driven at all address cycles of the transfer.

NOTE—Opcodes should be sampled at the end of an address cycle.

A[31:2] **Address Bus Lines**: Address bus lines are driven by the granted master early during an address cycle. The address identifies and selects the slave (via **SEL**), which is accessed by the bus operation, as well as the memory location in the slave's address space.

A[31:2] may remain undriven during a NOP operation.

NOTES
1—The number of address lines to be implemented depends on system address space requirements.
2—Addresses should be sampled at the end of an address cycle.

SELy **Slave Select**: This signal is generated by bus control from **OPC[3:0**] and **A[31:2]** information during an address cycle. At the end of an

address cycle only one **SEL** signal shall be active and only one slave shall be selected. A slave which is selected by its **SEL** signal handles the following data cycle.

SELy=0 Slave y is not selected for the current bus operation and shall remain idle**.**

SELy=1 Slave y is selected for the current bus operation. It shall register the address, opcode and read/write information at the end of the address (/data) cycle and shall start to perform the requested read or write operation.

SEL shall not be asserted by bus control:

— during an address cycle when **OPC[3:0**]=NOP or
— during an address/data cycle when **ACK[2:0]** =~(RDY | RDM).

NOTES
1—The number of **SEL** lines depends on the number of slave interfaces in a system.
2—Normally a single **SEL** signal is sufficient per slave.
3—**SEL** generation is based on a (partial) decoding of **A[31:2]**. The decoding can be done either centralised at the bus control or decentralised close to the slave.

D[31:0]

Data Bus Lines: Depending on the transfer direction of the bus operation, data bus lines are either driven by the granted master or by the selected slave early during a data cycle.

During byte and halfword bus operations data is transferred *right adjusted* via the low order bits of the data bus. In these cases the upper bits of the data bus remain either undriven or carry an undefined value.

NOTE—The width of the data bus depends on the system requirements. **D[7:0]** are mandatory, **D[31:8]** are optional.

ACK[2:0]

Acknowledge Code Lines: Acknowledge code bus lines are driven by the selected slave early during a data cycle or in case no slave has been selected in the previous bus cycle by bus control (error case).

Table 3—Acknowledge Codes

ACK[2:0]	Identifier	Description
000	WAT	**Wait**, bus cycle not completed
001	RDM	**Ready-More**, bus cycle completed Read—can deliver more data Write—can store more data
010	ERR	**Error**, last bus cycle aborted
011	RDY	**Ready**, bus cycle completed
100	RTR	**Retract**, last bus cycle aborted

Table 3—Acknowledge Codes

ACK[2:0]	Identifier	Description
110	SPT	**Split**, bus cycle aborted/completed
1x1	RSD	Reserved

The **ACK[2:0]** codes have the following meaning:

Wait

Slave forces bus control and the granted master to add another data cycle when it needs more time to perform the current bus operation.

Read operation— slave has not yet driven valid read data on the data bus lines; master shall repeat data reception in the added data cycle.

Write operation—slave has not yet read the data from the data bus; master shall re-send data during the added data cycle.

If the address of another bus operation was pipelined with the current data cycle, the master shall also repeat the address cycle.

Ready

Master request **successfully** performed and bus operation terminated in the current data cycle.

Read operation— slave has driven valid read data on the data bus lines; master can continue.

Write operation—slave has read the data from the data bus and stored at the requested location.

If **LOCK** was asserted during the corresponding address cycle, bus control shall inhibit bus arbitration and not grant another master in this data cycle.

Ready-More

Same meaning as **Ready**, but slave provides additional status information to the master:

Read operation— slave would be able to deliver more data to the master.

Write operation—slave is able to process further data from the master.

If **LOCK** was asserted during the corresponding address cycle, bus control shall inhibit bus arbitration and not grant another master in this data cycle.

NOTES—**Ready-More** has no special meaning to the PI-Bus protocol other than **Ready**. It is intended to be used e.g. by FIFO organised slaves to indicate the internal buffer fill status. Dedicated masters can make use of the **More** information.

Retract

Slave does not accept bus operation or cannot immediately complete it. Both, master and bus control, shall regard the bus operation as

aborted.

In the bus cycle following the **Retract** the master shall release all bus lines and abort the bus transfer. It is mandatory that the master again requests bus ownership to repeat the transfer, but not earlier than one bus cycle after **Retract** (resp. after two bus cycles if a block transfer is aborted).

If **LOCK** was asserted during the corresponding address cycle, bus control shall inhibit bus arbitration and not grant another master in this data cycle.

In case of a fixed length block transfer a slave may issue a **Retract** code only at the first bus operation.

NOTES
1—The optional **Retract** code is intended for use by slaves that may cause a significant amount of wait cycles to terminate the requested operation. **Retract** allows the slave to start an internal action without keeping the bus occupied until the action is completed. A master has to repeat the bus transfer until the slave terminates the bus operation with a **Ready** or **Error** status. Other bus masters have the chance to get bus ownership in between this polling sequence.
2—It is unspecified whether slave and master exchange any data during a **Retract** data cycle.

Error The selected slaves terminates the current bus operation and bus transfer with error status. In the bus cycle following **Error** the master and the slave shall abort their activities and release all bus lines.

NOTE—The handling of an error condition is left to the implementation.

Split The selected agent has accepted the read or write request from the master. The agent only acknowledges the request, but not the execution of the request. The slave will send the acknowledge of the execution in an own bus transfer. In case of a read, data will also be sent in that bus transfer.

NOTE—The optional **Split** acknowledge code is intended for future enhancements. A protocol for **Split** acknowledge handling has not yet been defined at the present stage. Master which are not supporting **Split** shall treat the **Split** acknowledge code same as **Error**.

2.6.4 Error Control Signals

TOUT **Timeout**: This signal is driven by bus control. **TOUT** may be asserted during a bus transfer if a slave device is not terminating a bus operation within a given (implementation dependent) time limit. An active **TOUT** causes any master or slave to stop driving any bused line, starting from the following bus cycle. An ongoing bus transfer is aborted.

TOUT=0	Normal operation
TOUT=1	Timeout, the active master and slave shall instantly abort their activity.

2.7 Recommendations for a PI-Bus implementation

Short bus transfers

Long bus transfers should be broken up into shorter ones. It is not intended that bus transfers can be interrupted despite in the case of fault conditions. Therefore, a long bus transfer may prevent a high priority master to access the bus.

One slave per bus transfer

In principle it is possible to address a different slave at every bus operation of a bus transfer. However, this may cause a conflict at the data and acknowledge lines if the shut off time of the drivers of one slave overlaps with the switch on time of the drivers of the other slave. To avoid this a no-operation shall be inserted at every time a change of slaves takes place.

NOTE—In the case of distributed address decoders, slaves also have to monitor the acknowledge lines to avoid that they drive to early if the before addressed slave performs wait cycles.

Module clustering

To avoid high bus loads simple slaves should be clustered and provided with single bus interface. Each slave may get its own select signal but operates on a common port for the other bus signal lines.

3. PI-Bus Operations

3.1 Introduction

The primary operation of the PI-Bus protocol is the bus operation. A bus operation is an exchange of a single data between a master and a slave. It is composed of at least one address cycle and one data cycle. Data wait cycles may be inserted between these two cycles if a master's read or write request cannot immediately be processed by a selected slave. Bus operations may be chained by a master and processed in sequence. Before a single bus operation or the first of chained bus operations can be started a master has to request bus ownership. Bus control then arbitrates which of the requesting master gets bus ownership. The request, arbitration and bus operation phases form together a complete bus transfer. In case of multiple chained bus operations, a bus transfer may also be referenced as block transfer.

3.2 PI-Bus Transfers

This section discusses various types of PI-Bus transfers. The symbols used in the following timing diagrams are explained below:


—Signal is not driven, but has a logic level due to the signal hold mechanism.


—Signal is driven, has a logic level, and may be evaluated.


—Signal may be driven but has a don't care value.


—Indicates how the assertion of a signal is related with other signals.

① —These numbers are referenced in the explanatory text for a diagram.

3.2.1 Single Data Read/Write Transfers

Figure 8 illustrates two PI-Bus transfers which are initiated by two different PI-Bus masters. Both transfers are single data bus transfers. The first one is a write operation and the second is a read operation. Both slaves are fast enough to handle reads and writes without data wait cycles.

① Master *a* receives an active **GNT** signal from bus control. This signal is the arbiter's response to an activated **REQ** signal from master *a*. Master *a* begins an address cycle by driving the signals **LOCK**, **READ**, **OPC[3:0]**=WDU, and **A[31:2]**. **LOCK**=0 signals to bus control that the current bus operation is the only one of the started transfer. ② At the end of the address cycle **READ**, **OPC[3:0]**, and **A[31:2]** are valid and shall be registered by that slave whose **SEL** signal is asserted.

In the following data cycle the master stops driving **LOCK**, **READ**, **OPC[3:0]**, and **A[31:2]**, puts write data on the **D[31:0]** lines and waits for the slave's processing status, which is indicated via the **ACK[2:0]** lines. ③ **ACK[2:0]**=RDY (or RDM) indicates that the write data had been processed by the slave.

ACK[2:0]=RDY (or RDM) **also informs bus control that another master may be granted at the next bus cycle**.

NOTE—In case the slave has more storage space for write data it may also respond with a **Ready-More** code (RDM). RDM doesn't have any influence on the protocol other than RDY!

In figure 8 no other master is requesting PI-Bus ownership. ④ Therefore, bus control asserts a default grant which is accepted by the granted master two bus cycles later at **CLK** ⌟. Because of its default grant capability the granted master immediately starts a read operation, indicated by **READ**=1. The default grant mechanism is more detailed explained in chapter 5.

.

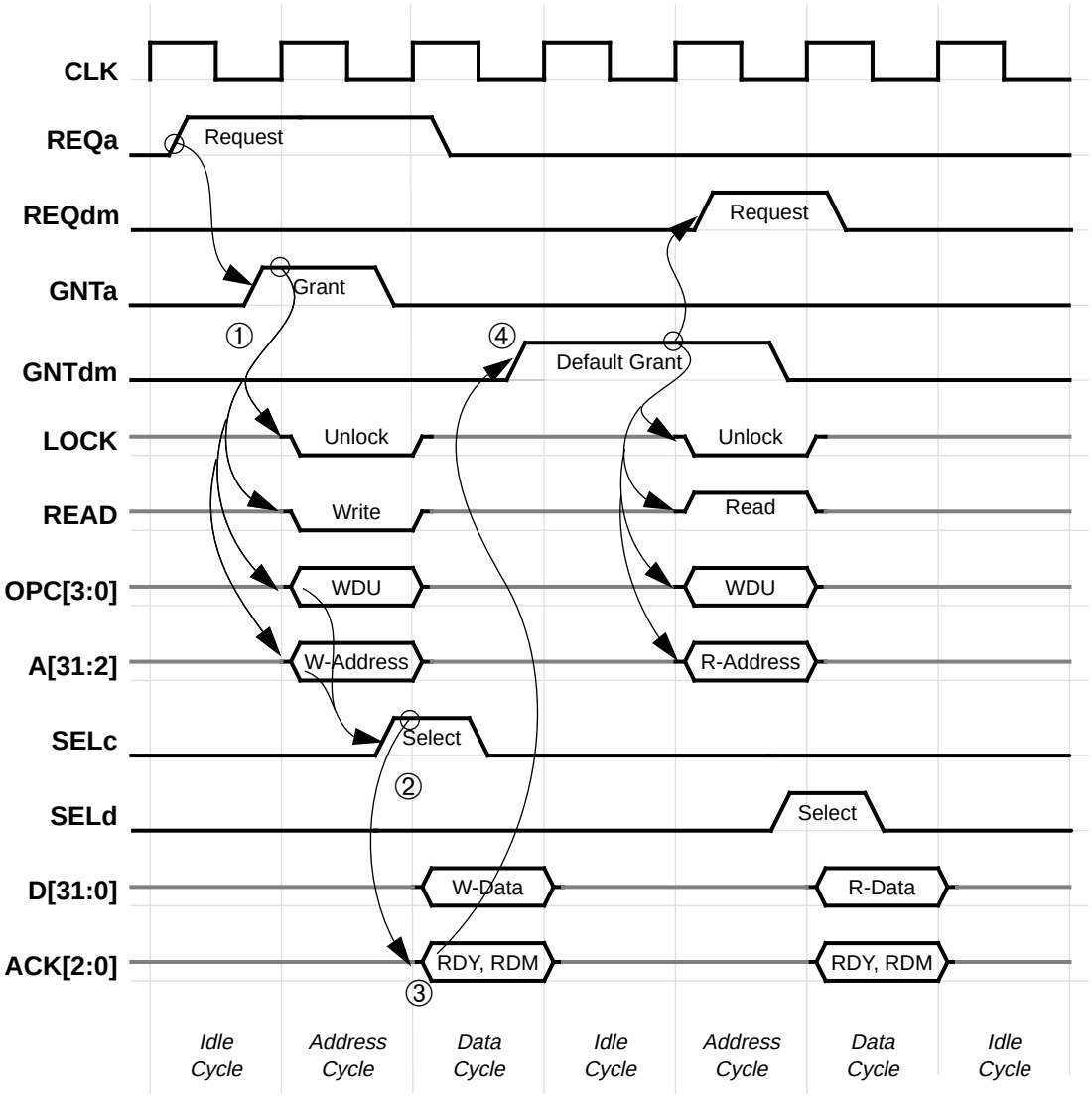


Figure 8 : Single data write and read bus transfers

Figure 9 shows write and read transfers with inserted wait cycles. **ACK[2:0]**=WAT informs the master and bus control, that the slave is not yet ready to store data in the case of a write or to deliver data in the case of a read. At writes the master shall re-send data in the following bus cycle. No **GNT** is generated in a data wait cycle although a bus request from master *b* is already pending.

① **ACK[2:0]**=RDY indicates that the data have been processed by the slave and master *b* is granted to start its bus transfer.

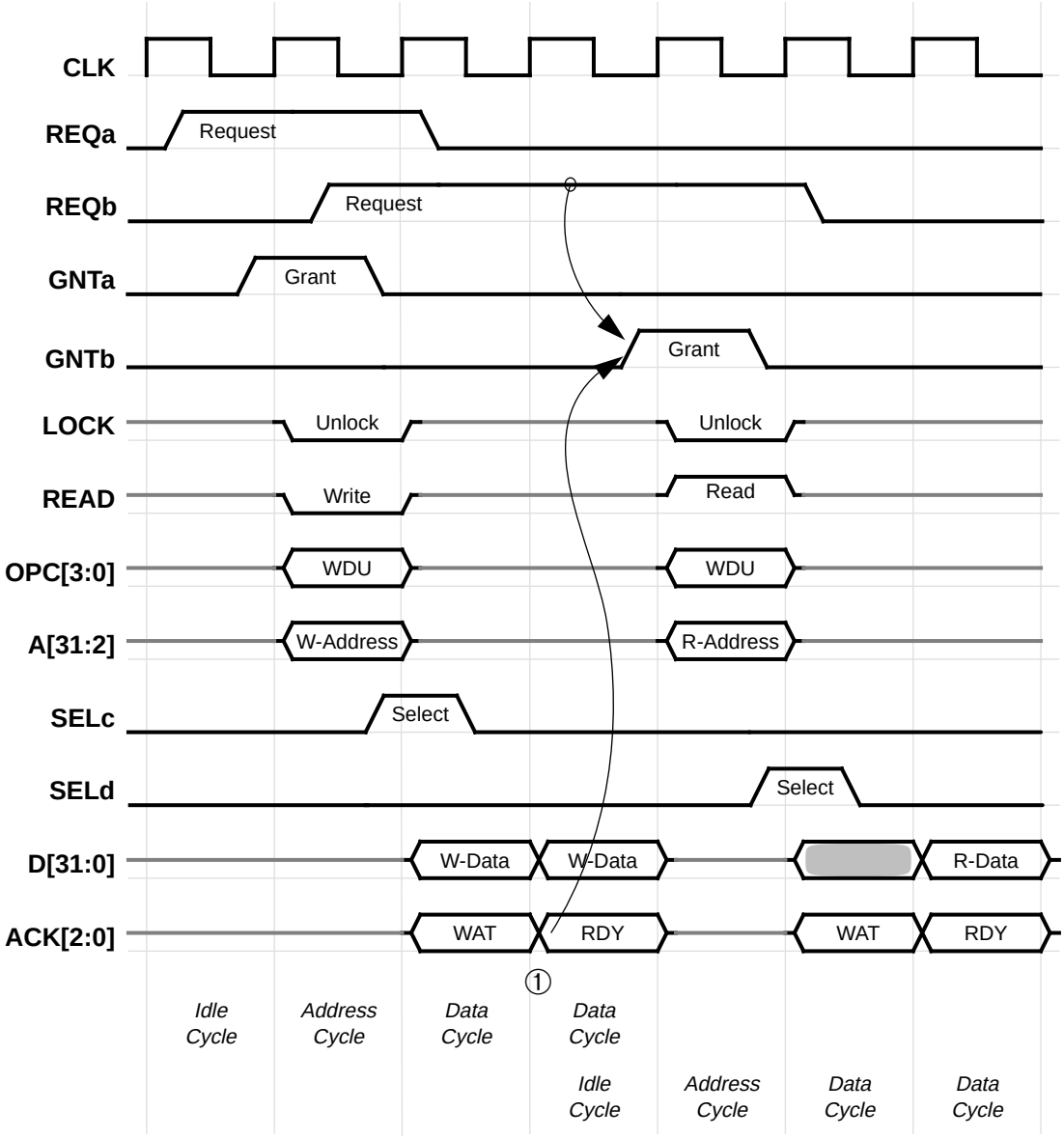


Figure 9—Single data write and read transfers with wait cycles

3.2.2 Block Transfers

Block transfers may have an undefined or defined number of chained (locked) bus operations. At block transfers multiple data are transferred between one dedicated master and normally one slave.

3.2.2.1 Transfers of Undefined Block Length

Figure 10 shows an example of a bus transfer with undefined block length, which is composed of three 32-bit read operations. The second read operation has one data wait cycle inserted.

① The block transfer starts when the master gets the grant signal. The master begins to drive **LOCK**, **READ**, **OPC[3:0]**=WDU, and **A[31:2]**. ② The address is within the allowed address range and the addressed slave gets an active **SEL** signal. ③ At the next bus operation **ACK[2:0]**=WAT indicates a *bus operation not completed* condition and no **SEL** signal shall be generated by bus control. The active master shall repeat the address cycle of the overlapping bus operation. ④ **ACK[2:0]**=RDY together with **LOCK**=0 in the corresponding address cycle signals that the transfer is finished and that a new **GNT** can be given.

In addition to a single data bus transfer, the block transfer protocol includes *address/data cycles*, where the data cycle of one bus operation is overlapped by the address cycle of the next bus operation. In the diagram of figure 10 three address/data cycles are shown.

NOTE—Instead of **ACK[2:0]**=RDY, a slave may also send RDM to indicate that more data can be processed.

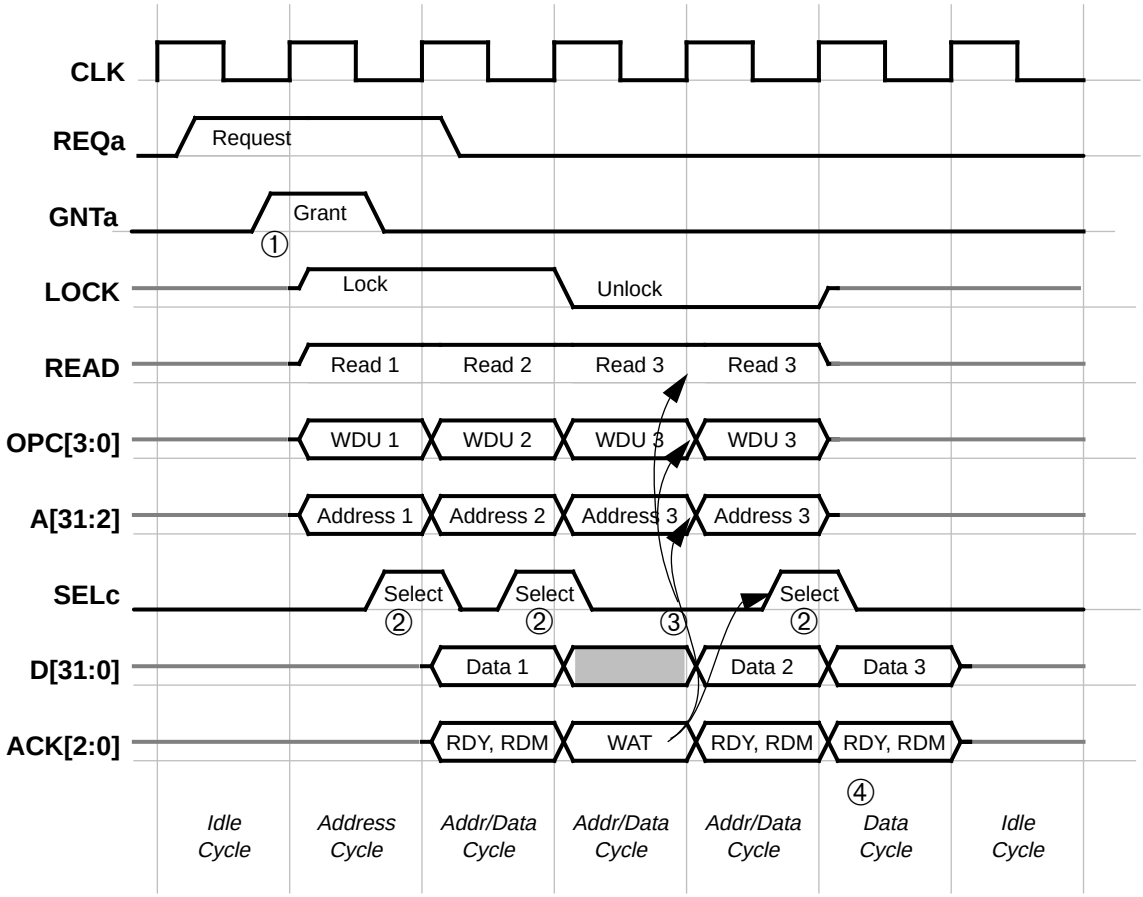


Figure 10—32-Bit undefined length block transfer

The following example in figure 11 illustrates how a FIFO memory at a slave can be emptied with the help of undefined length block transfers, the opcode NOP and the acknowledge codes RDM and RDY.

After every read operation, the granted master issues by default one or more NOP operation codes. The number of inserted NOP operations depends on the internal speed of the selected slave agent. The operation code which follows the NOP operation codes is influenced by the

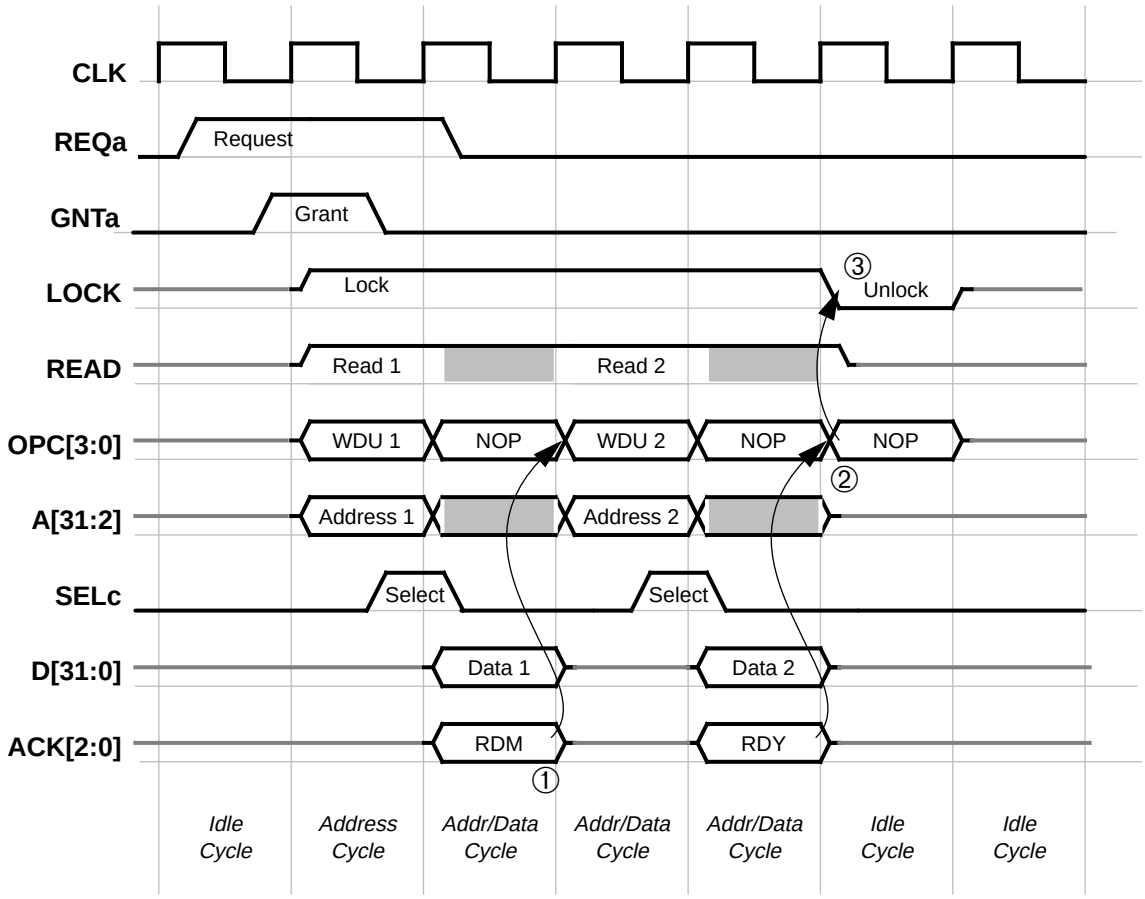


Figure 11—RDM/RDY-controlled emptying of a slave's FIFO

status of the acknowledge codes of the previous read operation. ① As long as **ACK[2:0]**=-RDM (Ready-More), **OPC[3:0]** is set to WDU. The slave issues RDM as long as its read-FIFO is not empty. ② With the last FIFO data the slave sends **ACK[2:0]**=RDY (Ready) which causes the master to switch **OPC[3:0]** from WDU to NOP. ③ At the same time the continuously asserted **LOCK** signal is reset.

NOTE
1—Fastest operation is achieved with low internal delays at the active master and slave.
2—The traditional approach requires to read the status information of a slave before every FIFO access.

In figure 12 an undefined length bus transfer is used to implement a read-modify-write operation. In this example, the master reads data from a slave, modifies it and writes it back again. **The master shall delay the write operation to avoid data bus contention between data the slave had driven and data the master drives**. A master delays bus operations via **OPC[3:0]**=NOP.

① The master begins with a read operation, which the slave terminates after one data wait cycle. ② Before the write operation takes place the master inserts one or more NOP operations. At least one NOP operation is required to avoid bus contention. ③ No slave is selected during NOP operations! The NOP operations allow the master to alter the read data. ④ The address cycle of the write operation is entered which writes the data back.

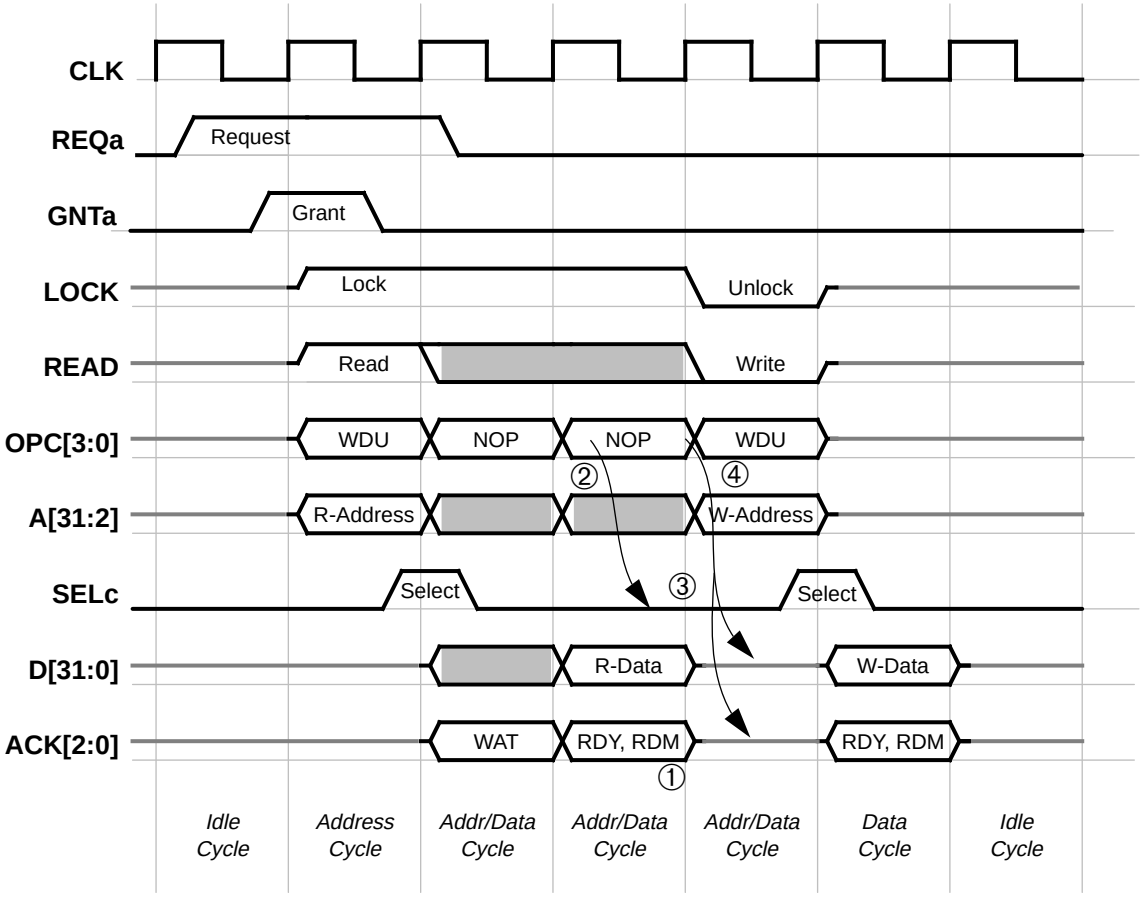


Figure 12—Undefined length block transfer used for a read-modify-write operation

NOTE—In principle, it is possible that a master accesses distinct slaves in a block transfer. However, to avoid bus contention it is necessary, that a NOP operation is inserted between bus operations which access two different slaves

3.2.2.2 Transfers of Defined Block Length

The PI-Bus protocol provides transfers of defined block length. This transfer type is intended for very fast transfers of data from consecutive memory locations. Therefore, it is well suited for e.g. cache line refills.

The block length of 2, 4, 8 or 16 words is transmitted with **OPC[3:0]** (WD2, WD4, WD8, or WD16). The active master shall repeat the opcode information in every address cycle of the defined length block transfer. To insert a delay the master may issue a NOP opcode and a no-operation bus cycle is inserted.

The handling of addresses during a defined length block transfer depends on the slave modules involved in the transfer. It is advised to use the defined length block transfer solely to access consecutive memory locations. Then, in the case of an intelligent slave it is sufficient to only provide the initial address of the memory block. An intelligent slave automatically assigns the higher addresses to the data sequence and performs the data access in the most efficient way and with highest speed. Such a slave just takes note of the first address of a transfer. A non-intelligent slave has to use every transmitted address to perform the transfer. It is very likely that a non-intelligent slave will introduce wait cycles at every bus operation.

NOTES
1—The access to consecutive memory locations can normally be performed much more efficiently than a random access. A bank-structured memory with four banks allows to access four data in one step.

2—A master module must know to which kind of slave module it is talking at defined length block transfers. An intelligent slave module will only require an initial address and perform sequential memory accesses automatically while a non-intelligent one will always require the whole address sequence.

In figure 13 a master starts a defined length block transfer of 4 words. The opcode WD4 is repeated during every bus cycle of the transfer. The address may or may not be incremented according to the type of the selected slave. In the diagram, master as well as slave delay the transfer via a NOP operation and a wait acknowledge code (WAT).

NOTE
3—In principle, the RDM acknowledge code doesn't make sense in a defined length block transfer although it may be returned instead of RDY by the selected slave.

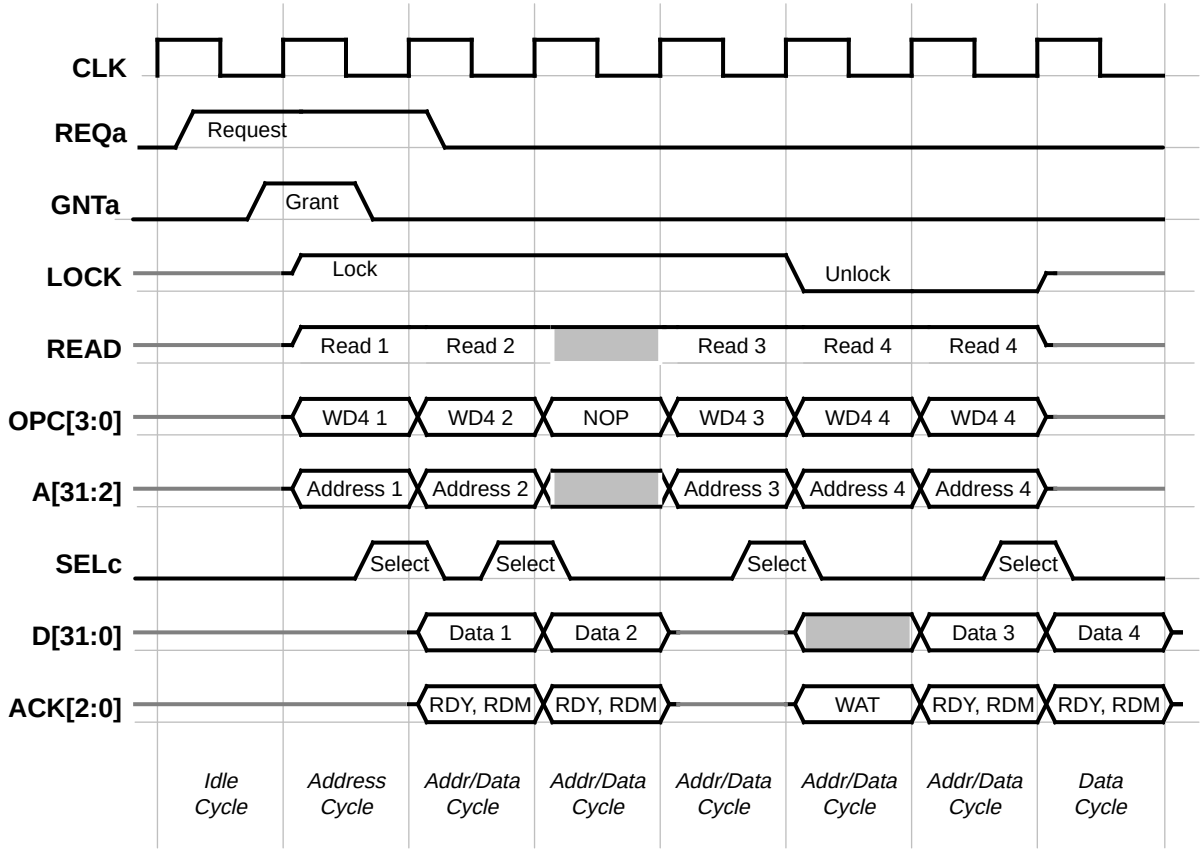


Figure 13—32-Bit defined length block transfer (4 words)

4. Transfer Termination

A PI-Bus transfer is initiated by a master and normally completed by the selected slave. The completion of read and write bus operations is indicated by the slave via the **ACK[2:0]** lines. Normal completion codes are the RDM and RDY code. Both acknowledge codes indicate that a bus operation has terminated successfully. The optional RDM code additionally informs the active master, that the slave is able to process more data.

A faulty bus operation and a transfer abort can be indicated by two other acknowledge codes: ERR and RTR. At these codes, the bus operation and the bus transfer shall be terminated immediately.

The additional **TOUT** signal allows to force the active master and slave off the bus. It is intended to be used if a slave does not terminate the current bus operation after a defined number of data wait cycles. This prevents a severe bus lock condition caused by a presumably blocked slave.

4.1 Bus Error Termination

The **Error** code (ERR) signals that the slave has detected some error condition and cannot handle the current bus operation. Possible reasons are an empty read FIFO, a wrong opcode (word access to a byte peripheral), or an address which doesn't belong to the slave address space. It is not defined whether data may come along with the ERR code. An implementation may use the data bus to transfer the error code in parallel to ERR.

Figure 14 gives an example for a block transfer of 4 words which is terminated by the slave with an ERR code. With the detection of the error code, master and slave stop driving their bus signals immediately in the next bus cycle. **Bus control may not grant another requesting master in an error terminated data cycle!** This enforces that an idle cycle is inserted before the next master start a transfer. The idle cycle is necessary at aborted block transfers to avoid a contention of the master driven bus lines between the aborted and the next transfer.

4.2 Retract Termination

The **Retract** code (RTR) signals that the slave module is busy and would need some time to handle the master request. The master shall retract from the bus in the next bus cycle to allow bus control to arbitrate new. A master which got the **Retract** code shall immediately request bus ownership to repeat the retracted bus transfer. The **Retract** timing and protocol corresponds to that of the bus error code (ERR). It is left to the implementation whether the slave takes write data or not. Read data is always undefined.

4.3 Split Termination

The **Split** code (SPT) is a type of ready condition. The timing and protocol is the same as in a normal bus cycle. The current bus operation and bus transfer is terminated normally but the requested operation is not yet performed at the addressed agent. The agent will respond later via an own transfer.

NOTE— The split operation is not yet implemented in the PI-Bus protocol. In a true split implementation the

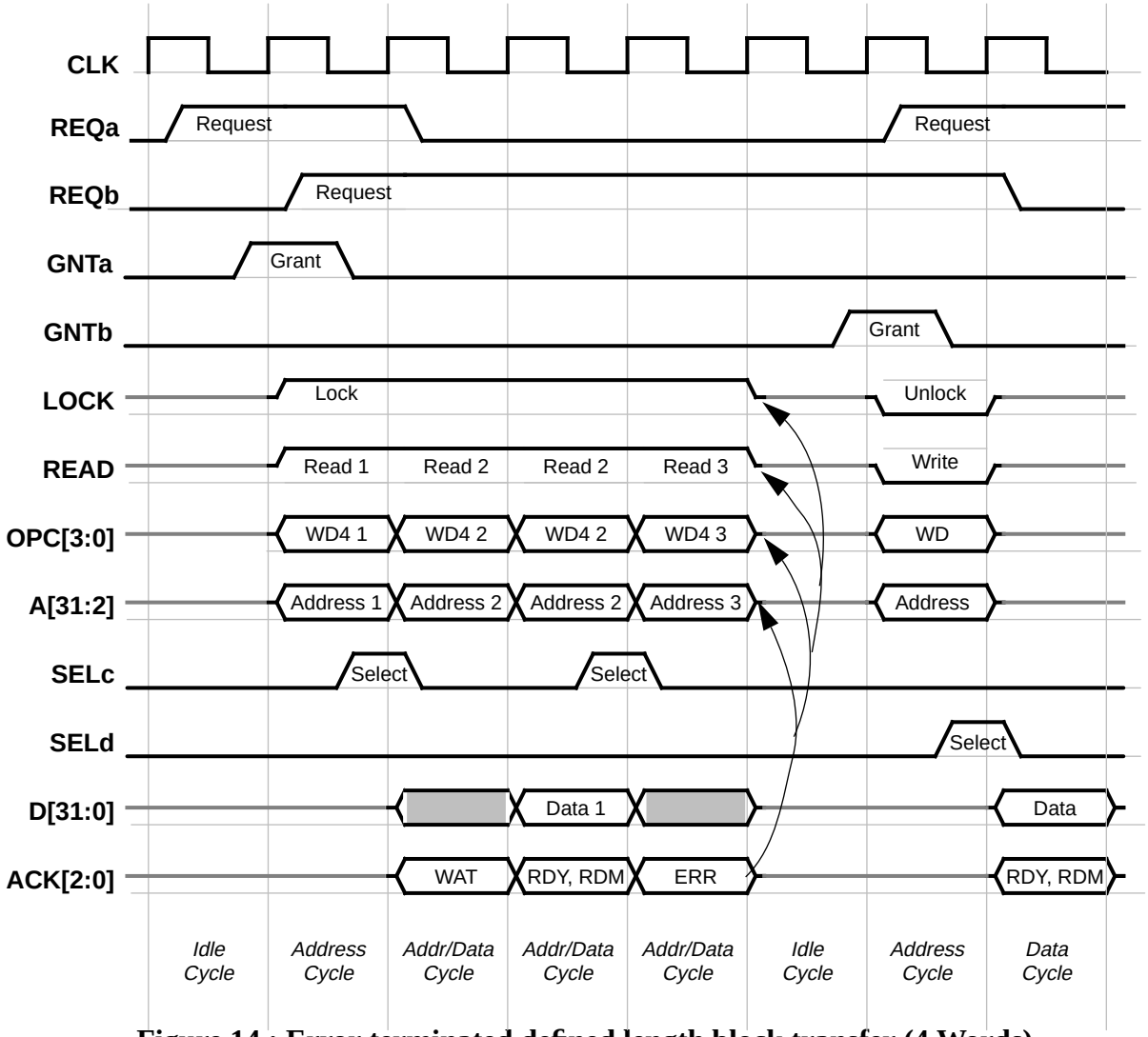


Figure 14 : Error terminated defined length block transfer (4 Words)

addressed slave will respond later by itself. This requires to transmit a master identifier, eventually with the help of additional signal lines. The transaction would also have to be tagged with a unique identifier. In the current version, a split acknowledge code should be treated as an error.

4.4 Timeout Termination

The timeout feature of the PI-Bus is a mechanism which avoids that a selected slave is driving the bus longer than an implementation defined number of bus cycles. When **TOUT** becomes asserted, the active master and slave interfaces shall stop driving their bus signals. Using **TOUT**, dead lock situations can be handled at the PI-Bus.

TOUT is a bus control generated signal. The timeout mechanism itself consists of a bus cycle counter which activates **TOUT** when a bus operation lasts more than a defined number of bus cycles. The number of bus cycles before timeout depends on the implementation of the PI-Bus.

Figure 15 shows the timing of the **TOUT** signal. If a block transfer is aborted by **TOUT** the master shall also stop driving the bus signals. Bus control shall not grant another master in a

timeout terminated data cycle. This is necessary to avoid the contention of master driven bus lines.

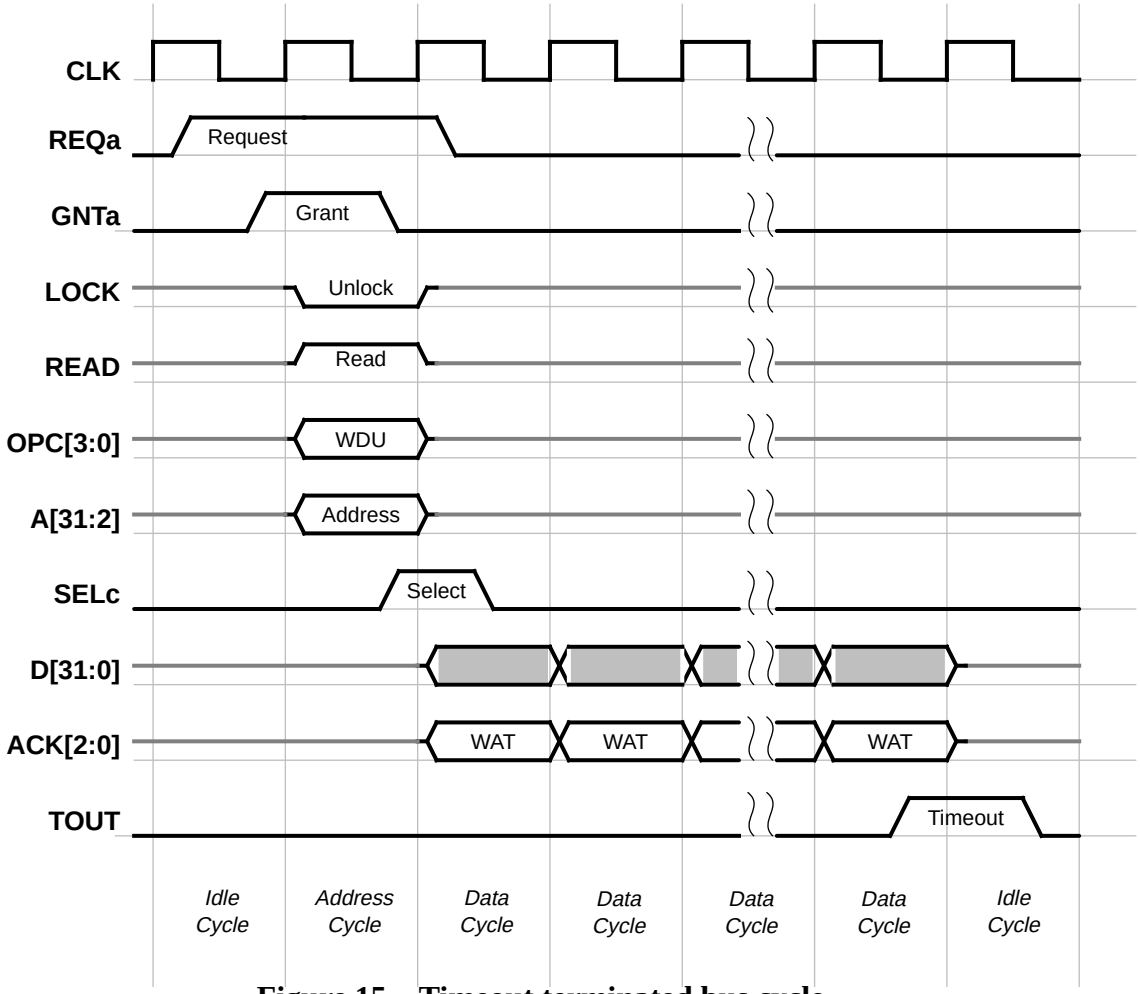


Figure 15—Timeout terminated bus cycle

5. Default Grant

The default grant feature is an option to the PI-Bus. Using this feature, one master interface, the so-called *default master*, always gets a grant from bus control if the bus is executing idle cycles and bus ownership is not requested by another master. The default **GNT** signal is activated even if the default master does not requests the PI-Bus.

A default master, which has been granted by default, may immediately execute a PI-Bus transfer. Via **REQ** the granted default master indicates to bus control in the bus cycle following the assertion of the default grant that it has started a bus transfer or not. If bus control detects an asserted **REQ** signal from a default granted master it shall wait for the end of the current default master controlled bus transfer before it may grant another requesting master.

Only one default master shall be granted at a time. The assignment of the default master is not fixed and may be changed dynamically.

Figure 15 shows a timing diagram of the default master mechanism. ① After a read bus operation of master *a* no further request is pending and the default **GNT** signal is given at to the default master. ② The default master starts a write operation and indicates via **REQ**=1 during the address cycle that it has accepted the grant. ③ After termination of the transfer no other master requests bus ownership and the default grant is asserted again. The granted default master does not start a bus transfer, so that the default grant is kept asserted until the next bus cycle. In this bus cycle master *a*, which is not the default master, requests the bus. ④ In response to the request, the default grant is released and the grant of master *a* is asserted. Master *a* begins its bus transfer in the next bus cycle.

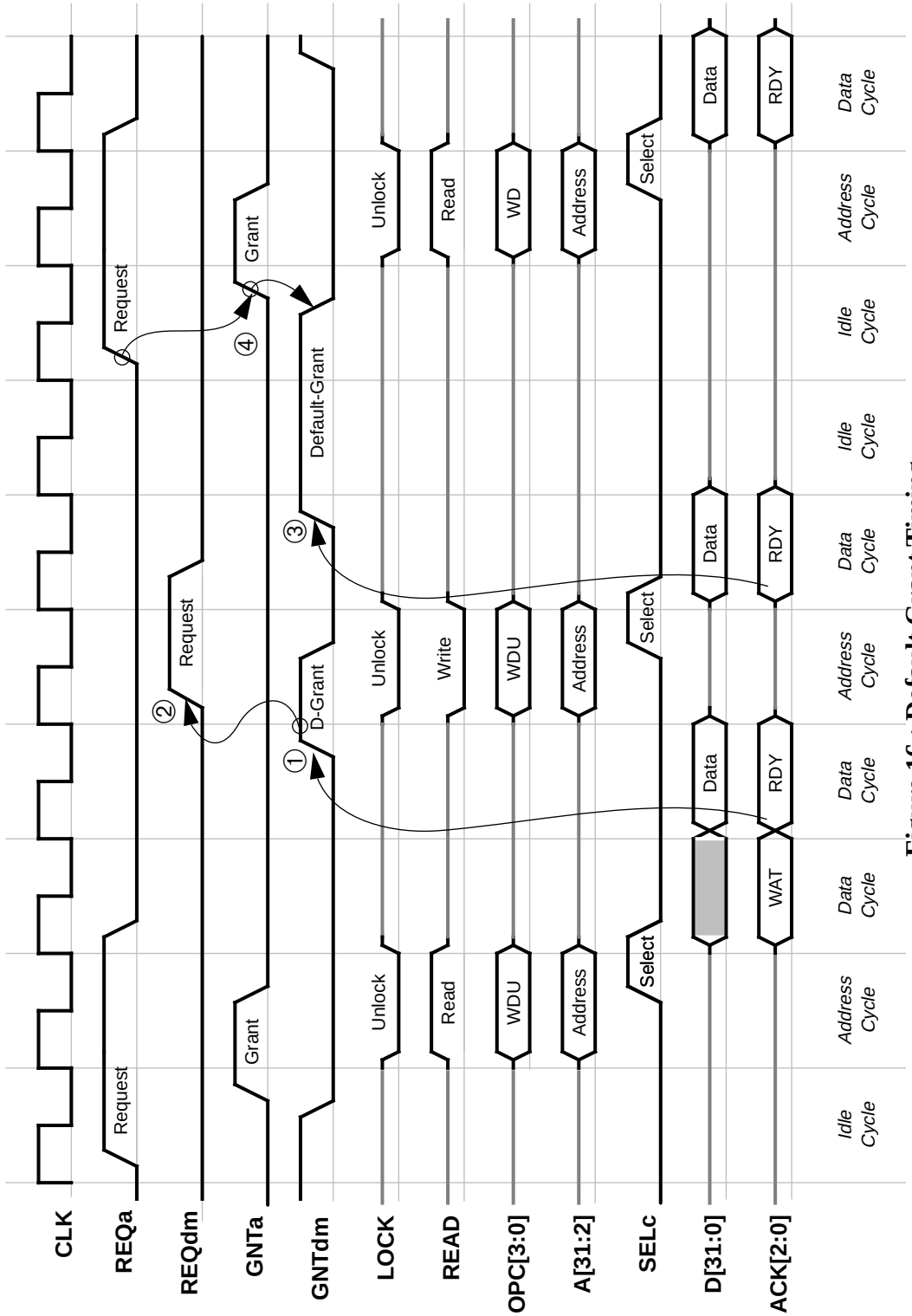


Figure 16 : Default Grant Timing