UM0484
User manual

STBus communication system concepts and definitions

Abstract

This document introduces the concepts and definitions referring to the STBus, the communication system (interconnect) developed for system-on-chip (SoC) applications and whose usage inside and outside ST is becoming wider and wider.

The basic terminology, the main concepts and the various available protocols are presented at a detailed but accessible level, to allow the reader to gain familiarity with the STBus and gain the background knowledge required to implement an STBus system.

Objectives

The objective of this document is to provide all the information required to understand the STBus. It covers the communication protocols and the interfaces.

Audience

This document addresses its audience as the interconnect architects, the interconnect designers, the application engineers and the project leaders.

The document can be used as a complete and concise reference about the STBus communication system by anyone starting architectural or design activities about the STBus.

Contents

1 Introduction

The STBus is a set of protocols, interfaces, primitives and architectures specifying an interconnect subsystem, versatile in terms of performance, architecture and implementation.

The STBus is the result of the evolution of the interconnect subsystem developed for microcontrollers dedicated to consumer applications such as, set top boxes, ATM networks, digital still cameras and others. Such an interconnect was born from the accumulation of ideas converging from different sources, such as the transputer (ST20), the Chameleon program (ST40, ST50), MPEG video processing and VCI (virtual component interface) organization.

Today the STBus is not only a communication system characterized by protocol, interfaces, transaction set and IPs, but also a technology allowing design and implementation of communication networks for SoCs with the support of a development environment including tools for system level design and architectural exploration, silicon design, physical implementation and verification.

1.1 STBus overview

1.1.1 Basic terminology

This section introduces some basic concepts that define a common language to be used when referring to an STBus based system.

For IPs connected to the STBus, two kinds of blocks are defined:

initiator — Any block able to generate traffic toward the interconnect in order to access the system resources (memory, peripherals and so on).

target — Any block being a resource for the system (memory, peripheral and so on), accessed by initiators.

In terms of data traffic over the interconnect, the following definitions are given:

transaction — An exchange of information between a generic initiator and a generic target.

cell — The basic amount of information that can be transferred within a clock cycle; the amount of information transmitted depends on the data bus size.

packet — A collection of cells, building an uninterruptible transfer; the number of cells building the packet depends on the operation size and the data bus size.

It is possible to build a more complex data structure using STBus interface signals in different ways:

chunk — A collection of packets linked together by the LOCK (LCK) signal (see *Section 2.1: Control signals on page 8*). Such a data structure allows locking the target until the transfer is completed. During the transfer no other initiators can access the target (uninterruptible).

message — A collection of chunks linked together by the TID[4] signal (see *Section 2.2: Transaction signals on page 10*). Such a data structure allows the transfer of a huge amount of data and exploits, when possible, proper routing schemes that optimize memory accesses and efficiency, particularly when accessing page based memories (such as DDR SDRAM).

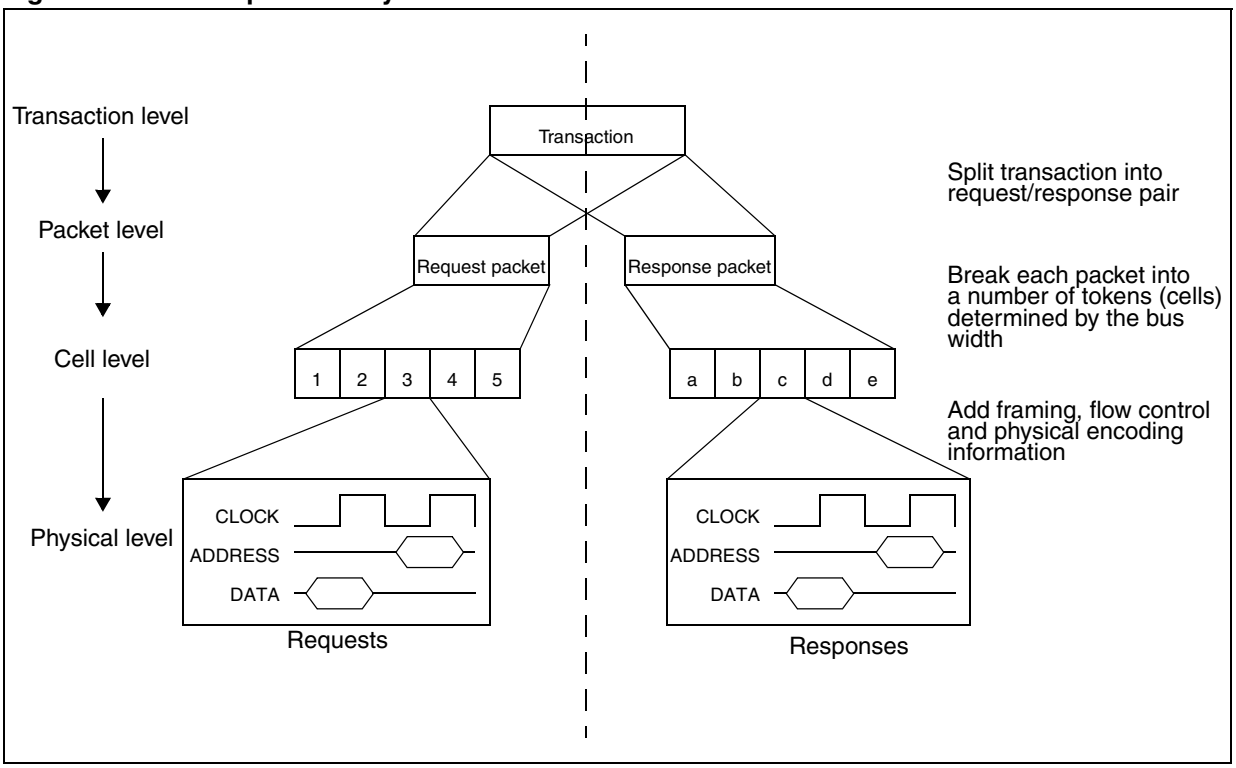
1.1.2 Protocol

Three different types of the STBus protocol exist, each having a different level of complexity in terms of both performance and implementation.

- **Type 1** is the simplest protocol and is intended to be used for peripherals registers access. No pipeline applies. It acts as an RG protocol. Load/store on 1, 2, 4, and 8 bytes are supported.
- **Type 2** adds pipeline features. It is equivalent to the “basic” RGV protocol. It supports all operation code for ordered transactions. The number of the requesting cells (that is, in a packet) is the same as the number of the response ones.
- **Type 3** is an advanced protocol implementing split transactions for high bandwidth requirements (high performance systems). It supports out of order executions. The packet response size might be different than the packet request size (the number of cells differs between request and response). The interfaces map the STBus transaction set on a physical set of wires defined by this interface.

The STBus protocol can be seen as organized in layers, as shown in *Figure 1*.

Figure 1. STBus protocol layers



1.1.3 Interfaces

There are two different types of interfaces: one between the initiator and the interconnect, the other between the interconnect and the target.

The difference between the interfaces consists of some signals that are used by one interface and not by the other one, and by some signals that are present in both the interfaces but whose behavior changes depending on the interface itself (for example, refer to the GRANT signal in *Section 2.1: Control signals on page 8*).

1.1.4 Components

The components, or building blocks, of an STBus communication system are:

- node
- write posting handler
- register decoder
- generic converter

The node is responsible for the arbitration and the routing of the transactions. The arbitration is performed by one or more arbiters, the routing is performed by the datapath through a set of multiplexers and demultiplexers, driven by the signals generated by the arbiter(s).

The write posting handler is responsible for the management of posted store transactions.

The register decoder is basically a node specific for type 1 IPs. It arbitrates among type 1 initiators, and allows access to type 1 targets, usually peripherals and registers banks.

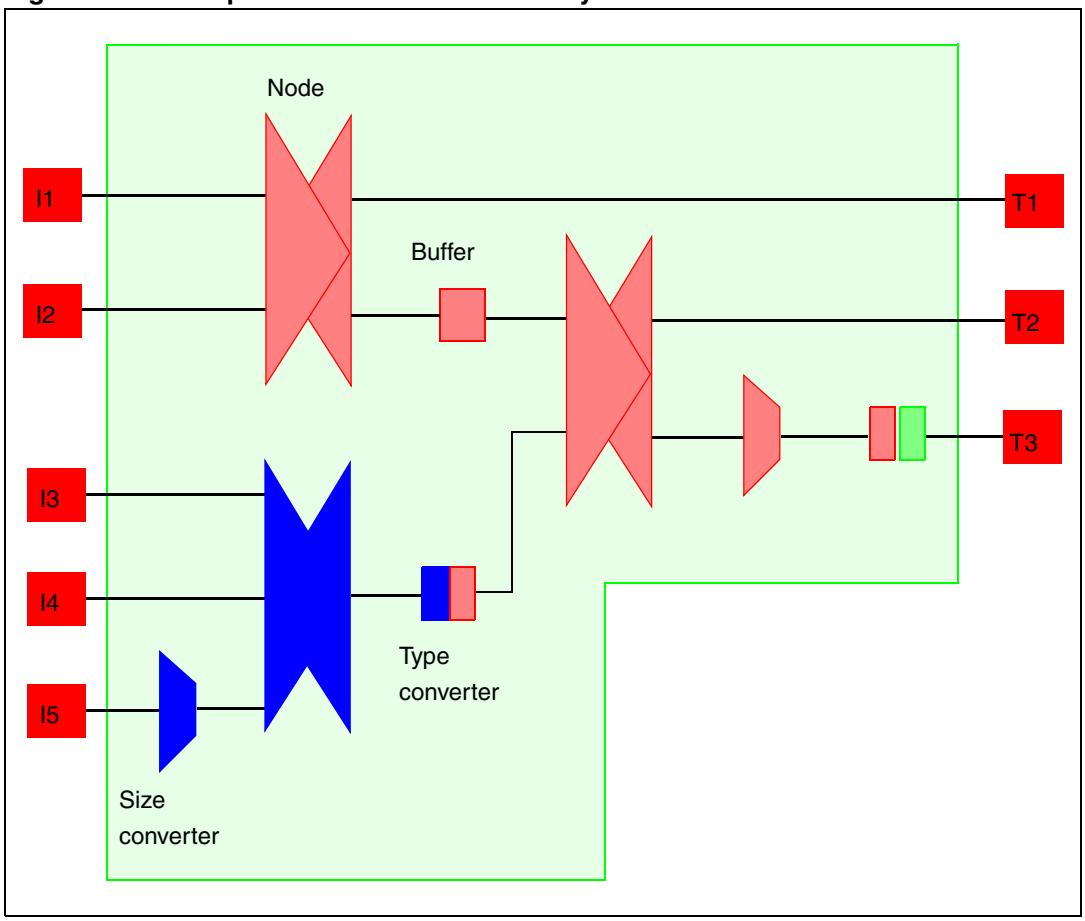
The generic converter can behave as one or more of the following blocks:

buffer	Used as retiming stage and is typically used to break critical paths, especially between blocks placed far from each other in the system floorplan.
size converter	Used to connect two blocks with different bus sizes to each other.
type converter	Used to connect two blocks following different STBus protocols to each other.
frequency converter	Used to connect two blocks working at different frequencies to each other, when the two clocks are completely uncorrelated (asynchronous mode) or when a phase relationship between them exists (semi-synchronous mode).

The generic converter can simultaneously perform a set of the operations (that is, size, type and frequency conversion) when behaving as in the blocks listed above.

Figure 2 shows an example of STBus interconnect where nodes, buffers, size and type converters are used. In *Figure 2* red blocks are type 3 blocks, blue blocks are type 2 blocks and green blocks are type 1 blocks.

Figure 2. Example of STBus interconnect system



2 STBus signals

This chapter contains a list of all signals and are defined in the context of the STBus protocols. Depending on the specific STBus protocol, not all the signals are required.

According to the protocol type, some signals are supported and some are not. In the tables in the following sections, supported signals are marked with 'S', unsupported signals are marked with 'U'. Among the supported signals, the mandatory and optional signals are highlighted in the sections dealing in detail with the three different protocols.

The STBus signals present at any STBus module interface can be grouped into three sets:

- control signals
- transaction signals
- service signals

2.1 Control signals

The control signals determine when a transaction starts, when it finishes, and if two or more consecutive transactions are linked together to build more complex transactions.

2.1.1 Initiator to target

Table 1. STBus control signals: initiator to target

Signal name	Description	Protocols
REQ (request)	The request is the signal used by the initiator interface to communicate that it wants to start a transaction. Once the request is asserted, all the signals associated with the packet to be transmitted must be kept constant until the grant signal (GNT) is received from the target interface, meaning the transaction has effectively started. The cell is actually transferred in correspondence to the clock rising edge, where both REQ and GNT are high.	Type 1 S Type 2 S Type 3 S
GNT (grant)	The grant is the signal with which a transaction is considered started after a request has been asserted. The GNT generated by a node as consequence of the arbitration is a "causal" grant (grant on request), while the grant generated by a target module is a "default" grant and is itself an information signal about the availability of the target to accept a new request. The cell is actually transferred in correspondence to the clock rising edge where both REQ and GNT are high.	Type 1 U Type 2 S Type 3 S
EOP (end of packet)	The end of packet is the signal marking the last cell of a currently transmitted packet. It is mainly used by the node to ensure a packet cannot be interrupted.	Type 1 S Type 2 S Type 3 S
LCK (lock)	The lock signal is used to link together two or more packets to build complex transactions, such as chunks and READMODIFYWRITE operations (see *Reserved operations on page 33*).	Type 1 U Type 2 S Type 3 S

2.1.2 Target to initiator

Table 2. STBus control signals: target to initiator

Signal name	Description	Protocols
R_REQ (response request)	The response request is the signal used by the target interface to communicate that it wants to complete a transaction. Once the R_REQ is asserted, all the signals associated with the response packet to be transmitted must be kept constant until R_GNT is received from the initiator interface, meaning the transaction has completed.	Type 1 S Type 2 S Type 3 S
R_GNT (response grant)	The response grant is the signal with which a transaction is considered complete after an R_REQ has been asserted. R_GNT generated by a node as a consequence of the response arbitration is a “causal” response grant. The R_GNT generated by a legacy module such as a converter is a “default” response grant which informs of the availability of the legacy module to accept a new response. Pure initiators don’t have the R_GNT signal, since they are supposed to initiate traffic generation provided that they are able to handle all the related responses.	Type 1 U Type 2 S Type 3 S
R_EOP (response end of packet	The response end of packet is the signal that marks the last cell of the currently transmitted response packet. It is mainly used by the node to ensure a response packet cannot be interrupted.	Type 1 U Type 2 S Type 3 S
R_LCK (response lock)	The response lock signal is used to link together two or more response packets during the transmission of complex transactions, such as responses to chunks.	Type 1 U Type 2 S Type 3 S

2.2 Transaction signals

2.2.1 Initiator to target

Table 3. STBus transaction signals: initiator to target

Signal name	Description	Protocols
ADD[31:n] (address with n = 0, 1, 2, 3, 4, 5 depending on the data bus size)	This is the address of the memory location which the initiator wants to access. The address of a transaction must be transaction-aligned. For example, an 8-byte operation can be addressed to 0x0, 0x8 and 0x10, but not to addresses 0x4 and 0xC.	Type 1 S Type 2 S Type 3 S
DATA[8*2^n-1:0] (data with n = 0, 1, 2, 3, 4, 5 depending on the data bus size)	This is the data the initiator wants to send to the target in case of store operations.	Type 1 S Type 2 S Type 3 S
BE[2^n-1:0] (byte enables with n = 0, 1, 2, 3, 4, 5 depending on the data bus size)	The byte enables signal defines which bytes within a cell are significant. For operations whose size is smaller than the data width, BE specifies the low order address bits. For example, with an LD1 in a 32-bit system, address bits [1:0] are implicitly encoded in the BE signal. In systems where BE signals are not defined, all cells are assumed to be valid.	Type 1 S Type 2 S Type 3 S
OPC[7:0] (opcode)	This signal defines the type of operation the initiator wants to perform. OPC is composed of three parts. Bits [3:0] indicate the operation type, for example, load/store/cache operation. Bits [6:4] specify the size of the operation, for example, the number of bytes the operation involves. Bit 7 is reserved and in multiplexed buses/systems is used to distinguish between request and response packets. The STBus interconnect is a non-multiplexed system so bit 7 must always be low.	Type 1 S Type 2 S Type 3 S
SRC[9:0] (source identifier)	SRC is used for two reasons: the lowest bits can be used by the initiator to identify possible internal sub processes, the highest bits are used by the interconnect to unambiguously identify the initiator itself.	Type 1 U Type 2 S Type 3 S

Table 3. STBus transaction signals: initiator to target (continued)

Signal name	Description	Protocols
TID[7:0] (transaction identifier)	The TID signal is used by the initiator to label a transaction with additional information. It is split into two fields: TID[3:0] uniqueness information, TID[7:4] enhancement information. The uniqueness information is used by type 3 initiators to unambiguously identify the transaction. This allows the correct response to be detected in the case of out of order traffic. Type 2 initiators don't have this capability. The enhancement information is used to enhance the processing of an operation without changing its functionality.The following fields are defined: – TID[4] : not end of message '0' = this transaction has no relationship to the next transaction (end of message) '1' = the next transaction is related to this transaction (member of a message) – TID[5] : not device access '0' = this store operation will not be posted '1' = this store operation may be posted (that is, the system may choose to return an early response, deleting the true response when it is returned to improve performance reducing the latency between grant and response) – TID[6] : store and forward '0' = each time a cell is received/assembled, it can be immediately propagated to the target interface '1' = first collect all the cells building a packet and then send the whole packet to the target interface – TID[7] : reserved The message information is used by the initiator to indicate that the system should attend to keep message components together. This can increase performance with access to a page-based memory such as the SDRAM. The write posting information is used by the initiator to indicate that the information in response to a write operation will be ignored so that a 'dummy' response can then be returned early. The store and forward information is used to drive the storage behavior to the buffers.	Type 1 U Type 2 S Type 3 S

Table 3. STBus transaction signals: initiator to target (continued)

Signal name	Description	Protocols
PRI[3:0] (priority)	The PRI signal labels the request packet with an urgency level which the interconnect may use to implement preferential arbitration. The priority is encoded into a 4-bit field (1111 = highest priority, 0000 = lowest priority).	Type 1 U Type 2 S Type 3 S
ATTR[15:0] (transaction attribute)	The ATTR signal is used by the initiator to label a transaction with an additional attribute field, according to the following description. ATTR[3:0] : protection information (PROT) ATTR[7:4] : cacheability (CACHE) ATTR[9:8] : addressing policy type (PTYPE) ATTR[15:10] : reserved for user-defined applications The following fields are defined: – PROT[0] : privilege level (0 = normal, 1 = privileged) – PROT[1] : security (0 = secure, 1 = non secure) – PROT[2] : data or instruction (0 = data, 1 = instruction) – PROT[3] : exclusive lock (0 = disabled, 1 = enabled) – CACHE[0] : bufferability (0 = non bufferable, 1 = bufferable) – CACHE[1] : cacheability (0 = non cacheable, 1 = cacheable) – CACHE[2] : read allocate (0 = not read allocate, 1 = read allocate) – CACHE[3] : write allocate (0 = not write allocate, 1 = write allocate) – PTYPE[1:0] : addressing policy (00 = fixed, 01 = incremental, 10 = wrapped, 11 = reserved) The explicit addressing POLICY TYPE (PTYPE) signal has been introduced to allow STBus type 3 initiators to generate asymmetric load/store commands to AMBA AHB/APB targets.	Type 1 U Type 2 U Type 3 S

2.2.2 Target to initiator

Table 4. STBus transaction signals: target to initiator

Signal name	Description	Protocols
R_DATA[8*2^n-1:0] (response data with n = 0, 1, 2, 3, 4, 5 depending on the data bus size)	R_DATA is the data read from memory at a specified location during a load operation.	Type 1 S Type 2 S Type 3 S
R_OPC[7:0] (response opcode)	The R_OPC signal is used by the target to label a response to an operation according to the following description: R_OPC[0] : operation status (0 = success, 1 = failure) R_OPC[1] : specifies whether the error has been originated by a target external to the interconnect (when 0) or, by the interconnect itself (when 1) as a consequence, for example, of wrong address or security violation. When R_OPC[0] = 0, R_OPC[1] is meaningless. R_OPC[2] : reserved R_OPC[3] : operation type (0 = write, 1 = read) R_OPC[6:4] : operation type (copy of OPC[6:4]) R_OPC[7] : fixed to 1(1). Type 1 modules implement only bit 0 of this field.	Type 1 S Type 2 S Type 3 S
R_SRC[9:0] (response source)	The R_SRC signal is a copy of the SRC signal associated to the respective request packet and is used by the interconnect to identify the initiator to which the response has to be routed. For an initiator, the lowest bits can be used to detect the internal subprocess to which the response is related.	Type 1 U Type 2 S Type 3 S
R_TID[7:0] (response transaction identifier)	The R_TID signal is a copy of the TID signal sent during the transmission of the request packet, and is used by type 3 initiators to detect the request packet to which the response is related.	Type 1 U Type 2 S Type 3 S

1. R_OPC[7] is reserved for systems which implement multiplexed request/response transports.

2.3 Service signals

Table 5. STBus service signals

Signal name	Description	Protocols
CLK (module clock)	All modules require a clock for the STBus interface. All STBus signals are synchronous and significant on the clock rising edge.	Type 1 S Type 2 S Type 3 S
RST_N (module reset)	All STBus modules must be reset, and during reset all STBus request signals have to be inactive. How the module is reset depends on the reset methodology used in the specific design flow. However, unless otherwise indicated, the reset should be assumed to be asynchronous and active low.	Type 1 S Type 2 S Type 3 S
SECURITY[n-1:0] (target security mode)	This signal is generated by the system security controller (if one is used). At most, n equals the number of type 3 targets accessible through the interconnect system. The signal states whether the target is secure or non-secure, there by affecting the way in which requests are routed to it through STBus nodes. For a given target, 0 = target secure, 1 = target unsecure.	Type 1 U Type 2 U Type 3 S
POWER_DOWN[n-1:0] (target power down status)	This signal is generated by the system power management controller (if one is used). At most, n equals the number of type 3 targets accessible through the interconnect system. The signal states whether the target is on or in power-down mode, there by affecting the routing of requests to it through STBus nodes. For a given target,: 0 = target in power up mode, 1 = target in power down mode.	Type 1 U Type 2 U Type 3 S

2.4 Test signals

The STBus interconnect is fully scannable for testability reasons. However the test signals do not appear explicitly at STBus components or interconnect system interfaces, since application of test methodology is left to the system integration process and is neither performed at block level nor at interconnect level.

The test signals to be added to STBus interface are the standard ones, as shown in *Table 6*.

Table 6. STBus test signals

Signal name	Description	Protocols
TST_SCANENABLE (scan test enable)	The STBus interconnect can be fully scanned for testability. The TST_SCANENABLE signal is used to enable the scan test operation mode.	Type 1 S Type 2 S Type 3 S
TST_SCANIN (scan test input)	This is the scan chain(s) input used during the scan test operation mode.	Type 1 S Type 2 S Type 3 S
TST_SCANOUT (scan test output)	This is the scan chain(s) output to be monitored during the scan test operation mode.	Type 1 S Type 2 S Type 3 S

3 STBus protocols

The STBus interface family contains a number of interface variants with differing performances and complexity costs. In selecting which type of STBus interface to use in a specific design, the designer should consider the module and system costs, and select the simplest interface with the appropriate features and data width for that system.

To ensure easy availability of third party support and IPs, the STBus interfaces and protocols are closely related to the industry standards being developed by the open virtual standard interface alliance (VSIA).

A general concept related to memory addressing is the concept of endianess. The endianess affects the way in which data is organized in memory. The data field is made up of a set of byte quantities, each byte being associated uniquely with a specific byte enable bit. Each byte is organized as a little endian bit quantity. The data field may contain multiple bytes, these are organized in a byte significant manner. This means modules accessed by operations which are equal to the interface width of that module do not depend on endianness. However, the addressing of sub-word quantities is dependent on the endianess of the module associated with that interface.

Table 7 shows an example of how sub-word quantities are interpreted by differing systems for a 4-byte system.

Table 7. Endianess mechanism

data \ byte lane	little endian 3	2	1	0	big endian 3	2	1	0
4 byte, address + 0	MSB	+2	+1	LSB	MSB	+1	+2	LSB
2 byte address + 0			MSB	LSB	MSB	LSB		
2 byte address + 2	MSB	LSB					MSB	LSB
1 byte[1], address + 0				LSB	MSB			
1 byte, address + 1			LSB			MSB		
1 byte, address + 2		LSB					MSB	
1 byte, address + 3	LSB							MSB

1. For a single byte quantity the MSB and LSB are equivalent.

The main difference between the two addressing modes is the way in which the bytes of a word are interpreted within the memory locations: in little endian, a data word is accessed starting from the least significant byte (LSB) and proceeding toward the most significant byte (MSB). While in big endian the access starts from the most significant byte and proceeds toward the least significant byte. This means that, for a write operation to a memory location, the bytes building a data word are swapped within the data word.

For example, if an IP wants to write the four bytes B1, B2, B3 and B4 starting from address 0x0 (that is, B1 at 0x0, B2 at 0x1, B3 at 0x2 and B4 at 0x3) it has to build the following 4-byte word in little endian:

B4B3B2B1

and the following one in big endian:

B1B2B3B4.

The following sections introduce the three different STBus protocols: type 1 (peripheral), type 2 (basic) and type 3 (advanced).

3.1 Type 1 protocol (peripheral)

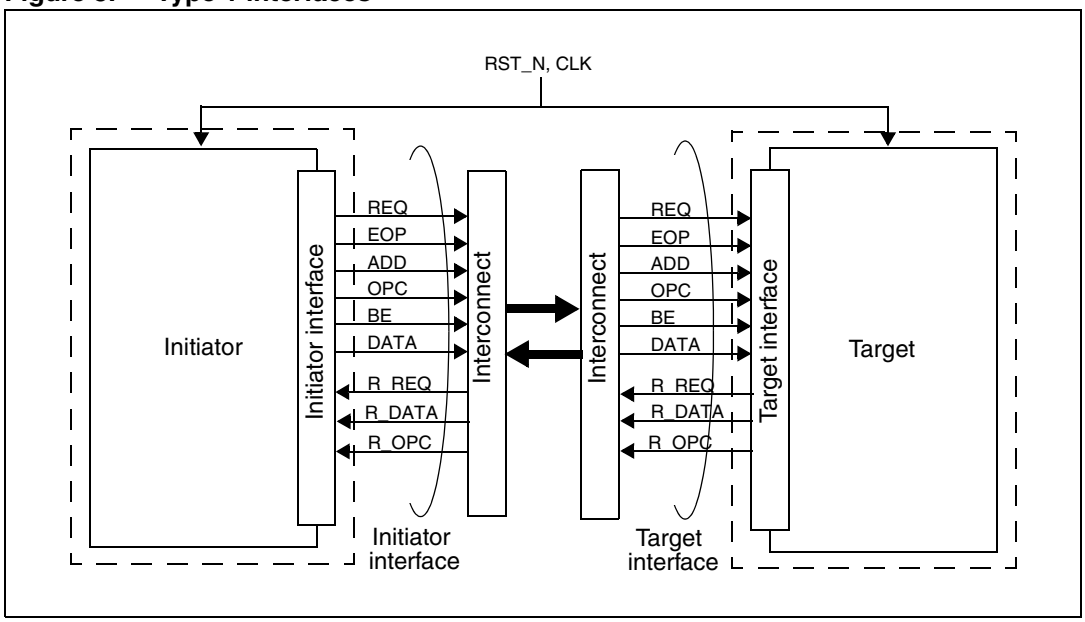
The type 1, or peripheral, STBus interface is the simplest of the STBus family. It is targeted at modules which require a simple, low complexity, medium data rate communication path with the rest of the system.

This typically includes standalone modules such as general purpose input/outputs (GPIOs) and UARTS, or modules which require independent control interfaces in addition to their main memory interface, or internal interfaces which do not require pipelines.

The supported operation subset is targeted at simple memory devices and supports read and write operations of up to 8 bytes. The interface is organized to minimize the need for the target device to interpret the operation, allowing the correct operation to be achieved by handling larger data quantities as if they were a series of smaller accesses.

3.1.1 Type 1 interface definition

Figure 3. Type 1 interfaces



The type 1 interface is a simple handshake interface which supports a limited set of operations. These operations are mapped onto a packet containing one or more cells at the interface.

Each request cell contains information on the operation type (OPC), the position of the last cell in the operation (EOP), the address of the operation (ADD) and the associated data (DATA) in case of write to memory. This cell is passed to the target, which indicates it has accepted the cell by asserting a handshake (R_REQ) and passes back a response.This response is also a packet containing a number of cells, each cell containing data (R_DATA) and optionally error information (R_OPC).

All type 1 devices are required to instantiate all mandatory signals, however simple devices may choose to only use a subset of these signals, leaving unused signals existing at the interface. These unused signals may not be connected if the functionality associated with the signal is not required.

As most modules are expected to be reused in large SOC systems with complex software, it is recommended that modules should implement a very basic error model which allows information on an incorrect operation to be passed back to debug software and helps build robust systems. The R_OPC field is an optional signal which may be used to achieve this. It is typically used to indicate that a specific operation is not supported or, that accesses to some address locations within this device are not allowed. The system (as a whole) is able to use this information to indicate a potential problem. Devices which do not implement an error module are always assumed to do the operations correctly.

3.1.2 Type 1 signal and timing

The signals given within *Table 8* are defined for type 1 initiators and targets. Signals which are mandatory for an interface are marked with 'M', signals which are optional and should only be implemented if required are marked with 'O'.

Table 8. Type 1 signals and timings

Signal group	Full name	Signal name	Direction	Initiator		Target	
				Timing	Type	Timing	Type
Request flow control and framing	Request	REQ	init to targ	early(1)	M	late(2)	M
	End of packet(3)	EOP	init to targ	early	O	late	O
Request contents	Opcode	OPC[2:0]	init to targ	early	M	late	M
		OPC[3](4)	init to targ	early	O	late	M
	Address	ADD[31:size(5)]	init to targ	early	M	late	M
	Byte enable(6)	BE[(2^{size}-1):0]	init to targ	early	O	late	M
	Data(7)	DATA[(8 * 2^{size}-1):0]	init to targ	early	O	late	O
Response flow control	Response request	R_REQ	targ to init	late	M	early	M
Response content	Response opcode	R_OPC	targ to init	late	O	early	M
	Response data(8)	R_DATA[(8 * 2^{size}-1):0]	targ to init	late	O	early	O

1. Early is defined as being in the first 20% of the clock cycle.
2. Late is defined as being in the first 80% of the clock cycle.
3. Initiator may not use it if generating single-cell packets only, target may not implement it only if its data size is 8 bytes.
4. Initiator may not use it since it has no defined meaning, target must implement if for reusability reasons.
5. Size defines the width of the interface, it may take a value between 0 and 3 and corresponds to interface widths of 1, 2, 4 or 8 bytes (8, 16, 32 or 64 bits).
6. Initiator may not use it if accessing entire words only.
7. Not used if initiator/target is read-only.
8. Not used if initiator/target is write-only.

Note: The STBus interconnect top level and building blocks always have all the signals at each interface, only external initiator and target IPs can use the optional signals.

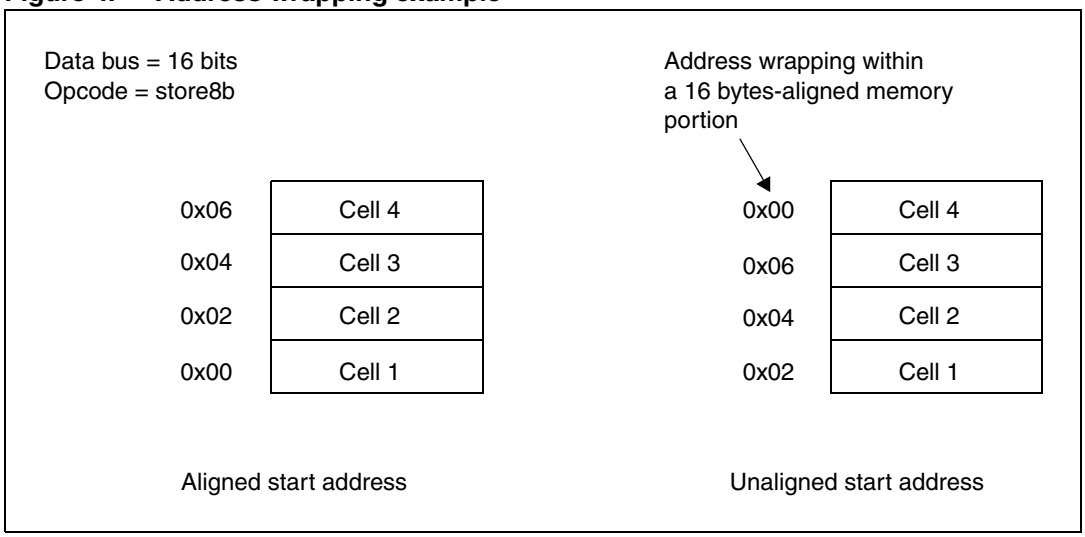
3.1.3 Address usage

As far as the address of a packet is concerned, the following rule applies: the packet address must be opcode aligned, that is, it has to be a multiple of the number of bytes to be transferred within the packet. This means that:

- for 1-byte operations valid addresses are 0x0, 0x1, 0x2, 0x3 and so on
- for 2-byte operations valid addresses are 0x0, 0x2, 0x4, 0x6 and so on
- for 4-byte operations valid addresses are 0x0, 0x4, 0x8, 0xC and so on
- for 8-byte operations valid addresses are 0x0, 0x8, 0x10, 0x18 and so on

If the start address of a packet is not transaction aligned, then a wrapping of the address has to be implemented, in order to keep the cells of the packet within the allowed memory portion aligned to the transaction. *Figure 4* shows the concept of address wrapping through an example.

Figure 4. Address wrapping example



Note: In type 1 and type 2 systems, the address wrapping for misaligned start addresses has to be performed by the initiators. In type 3 systems, the address wrapping can be performed by the targets as for store operations the address can be constant within a packet and for loads only one cell is generated.

3.1.4 Byte enables usage

Byte enables is a signal (BE), whose size is equal to the number of bytes the data bus is composed of and may be defined in two ways:

- when the size of the operation (in bytes) is smaller than the data bus size (in bytes) the BE signal represents the lowest part of the address and must obey the addressing rule “the packet address must be opcode aligned”
- when the size of the operation is bigger than the data bus size, BE is simply a mask specifying which bytes of the overall word are affected by the operation

To clarify the second definition: consider a store of a 2-byte operation (STORE2B) in a 32-bit (4 bytes) context. The address is a 30-bit signal (ADD[31:2]) and BE is a 4-bit signal (BE[3:0]).

To write 2 bytes starting from the address 0, use:

ADD = 0x0000 0000

BE = 0011

Only bytes 0 and 1 are written. To write 2 bytes starting from the address 2, use:

ADD = 0x0000 0000

BE = 0xC (0b1100)

Only bytes 2 and 3 are written.

Since the STORE2B operation must obey the rule regarding transaction alignment, valid addresses for a 2-byte operation are addresses that are 2-byte aligned (0x0, 0x2, 0x4, 0x6, 0x8 and so on). Addresses that are 1-byte aligned (0x1, 0x3, 0x5, 0x7 and so on) are not legal for this operation.

This implies that the BE signal cannot be “1001” or “0110” for this operation. The same applies for other bus and opcode sizes.

3.1.5 Opcode usage

The encoding used on the peripheral interface is given in *Table 9*.

Table 9. Load/store opcode encoding

Operation	bit[3]	bit[2:1]	bit[0]
load byte	0	00	1
load two bytes	0	01	1
load four bytes	0	10	1
load eight bytes	0	11	1
store byte	0	00	0
store two bytes	0	01	0
store four bytes	0	10	0
store eight bytes	0	11	0
reserved	1	--	1

The encoding is not generally interpreted by the interconnect or transport medium. This information remains constant across all the cells associated with this operation. Large (greater than 8 bytes) and compound operations are not supported on the peripheral interface.

If an unsupported operation is presented to a target then, if possible, the target device should indicate an error to the system. Irrespective of error support the target must always return a response.

3.1.6 Response opcode usage

The response opcode (R_OPC) field is used by the target to give information about the status of the operation. When the operation is successfully completed, the R_OPC signal has the value 0. When the operation fails, either because of a wrong address or an unsupported opcode, the R_OPC signal has the value 1. Some type 1 targets may assert the R_OPC when there is a write attempt is performed on read-only registers.

3.1.7 Basic transaction description

LOAD m bytes

Abbreviation	LD1, LD2, LD4, LD8
Definition	Read a single aligned word of m bytes from the target to the initiator. Valid sizes for m are defined to be 2^n where n is an integer in the range 0 to 3.
Qualifiers	ADD[31:n]: the address of the word to be accessed. BE[n-1:0]: the byte enable indicates which bytes within the word are significant. R_DATA [8 x 2^n-1:0]: data to be transferred, the significance of bytes within this field is inferred from the byte enable information. R_OPC: result of operation.
Comments	Load implements a read operation between the initiator and the target.

STORE m bytes

Abbreviation	ST1, ST2, ST4, ST8
Definition	Write a single aligned word of m bytes from the initiator to the target, overwriting the location at that address with the data transferred. Valid sizes for m are defined to be 2^n where n is an integer in the range 0 to 3.
Qualifiers	ADD[31:n]: the address of the word to be accessed. BE[n-1: 0]: the byte enable indicates which bytes within the word are significant. DATA[8 x 2^n-1:0]: data to be transferred.
Comments	Store implements a write operation between an initiator and a target.

Unsupported operation

Definition	Return a response on reception of the operation.
Comments	The target device should always return an error if possible if presented with an operation which is reserved. This allows the operation set to be extended for future systems without making existing IPs redundant. The response will always contain one response cell for each request cell received.

3.1.8 Examples

Figure 5. Single cell operation

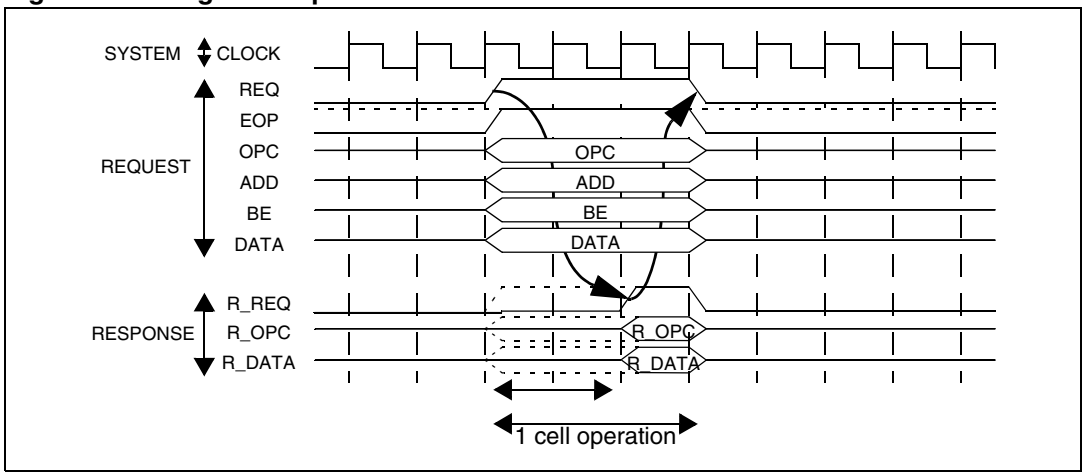


Figure 6. Two cells operation

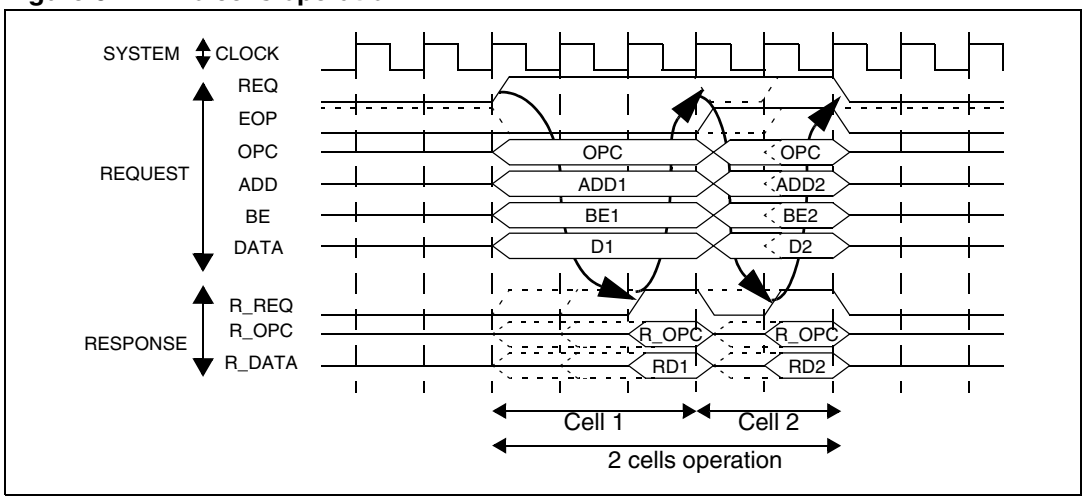


Figure 7. Four cells operation

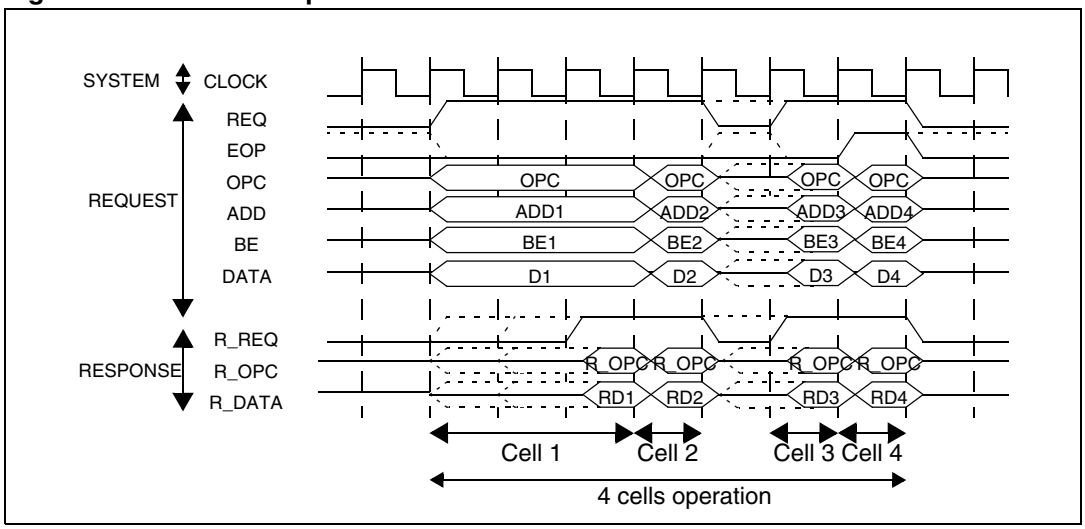
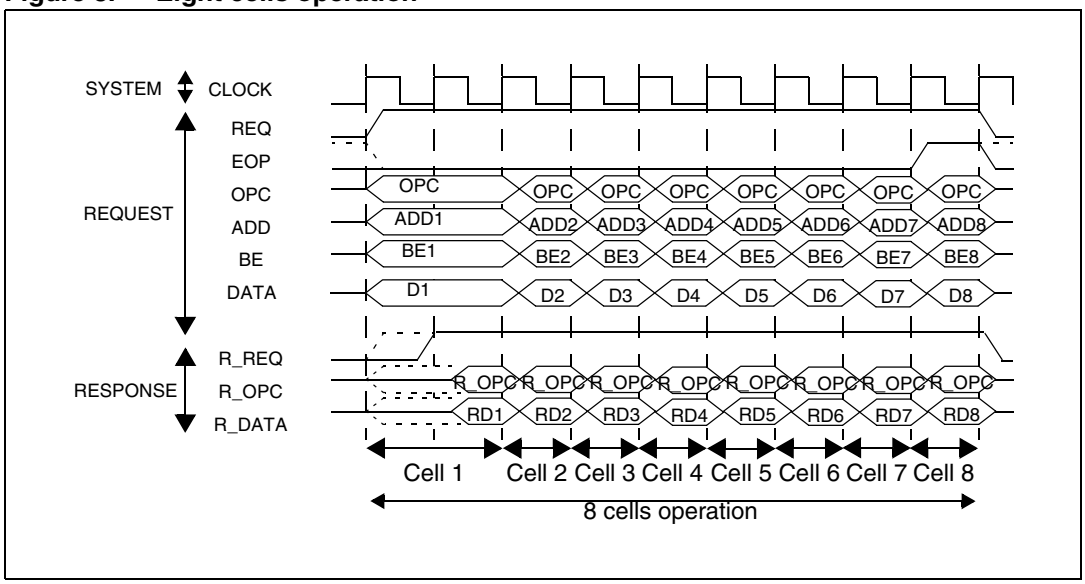


Figure 8. Eight cells operation



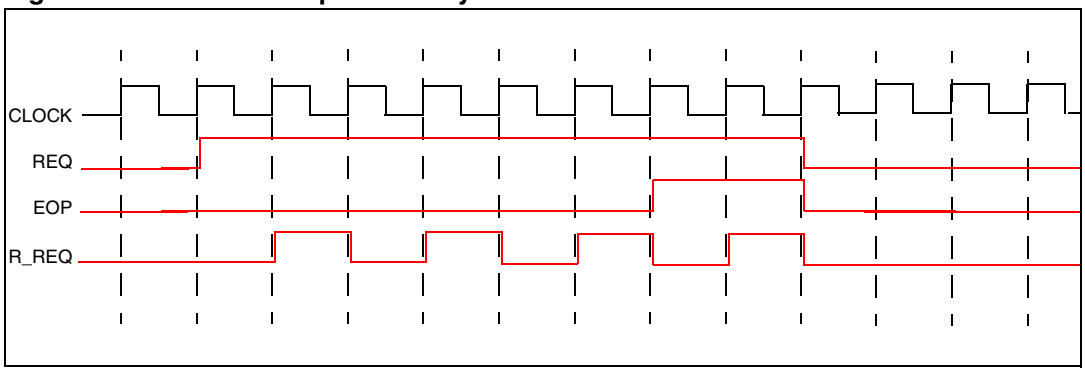
3.1.9 Remarks on response request delay

The following four example behavior types are theoretically possible for a multi-cell type 1 transaction consisting of the transmission of a packet composed of four cells.

Example 1

All R_REQ coming with one (or more) cycles of delay with respect to the REQ for each cell of the packet.

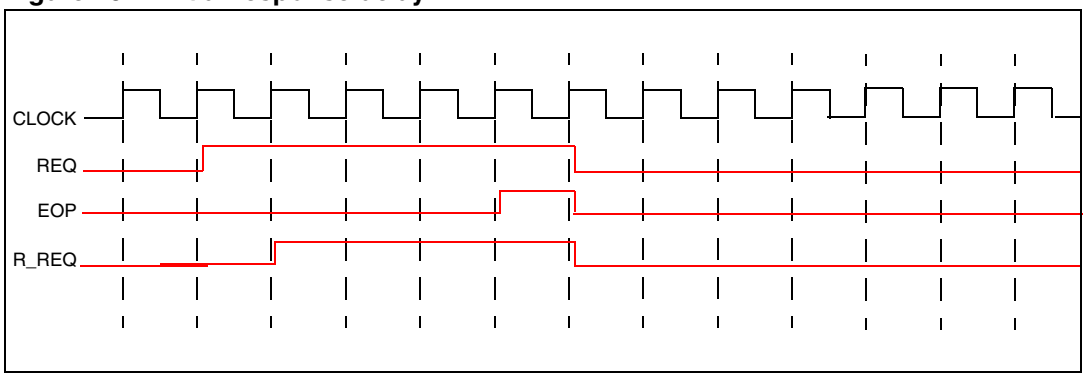
Figure 9. Constant response delay



Example 2

First R_REQ coming with respect to one cycle of delay with respect to the REQ, the others coming in the same cycle as the REQ.

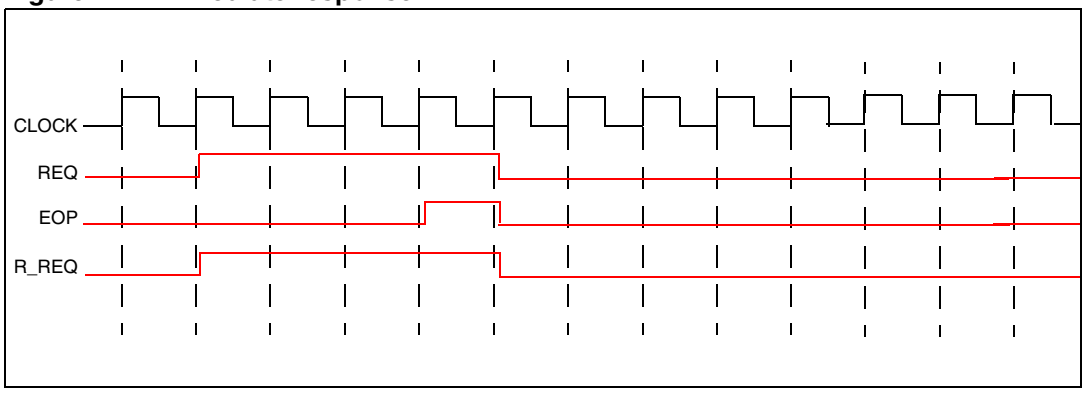
Figure 10. Initial response delay



Example 3

R_REQ coming in the same cycle as the REQ for all the cells of the packet.

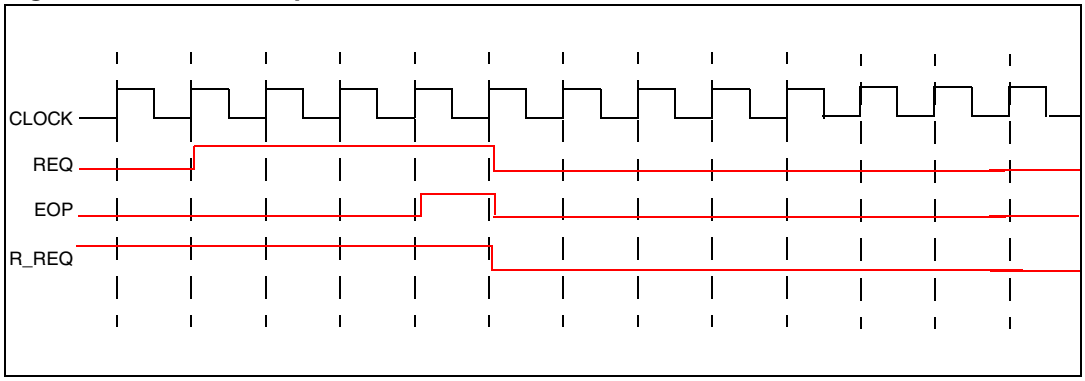
Figure 11. Immediate response



Example 4

R_REQ coming before the REQ (default grant behavior, see *Section 3.2.4 on page 30*).

Figure 12. Default response



Currently, the T3/T1 and T2/T1 converters and the register decoder work in the example behavior type shown in *Example 1*, but have not been verified with the other examples. They might work with the example behavior type shown in *Example 2*, but it is unlikely that they will work with the behavior types shown in *Example 3* and *Example 4*.

The T1/T3 converter works with the example shown in *Example 2*.

Most of the IP having a T1 interface work with the example shown in *Example 1*.

In terms of protocol specification, *Example 3* is a violation of the rule stating that there cannot be an R_REQ without a REQ, so *Example 3* is not allowed in type 1 protocol.

It is also true that, for timing reasons, *Example 2* is unlikely, even if it could be useful for slow peripherals (so slow that timing is easily met).

The example shown in *Example 1* is the most likely and currently the most widely supported, but is inefficient. The example shown in *Example 2* represents the optimum solution in cases of multi-cell packet operations. So the examples shown in *Example 1* and *Example 2* are the only examples allowed.

Note: It is possible that an incompatibility among IPs working only following the example shown in Example 1, and IPs working following the example shown in Example 2, must be solved at hardware level.

3.2 Type 2 protocol (basic)

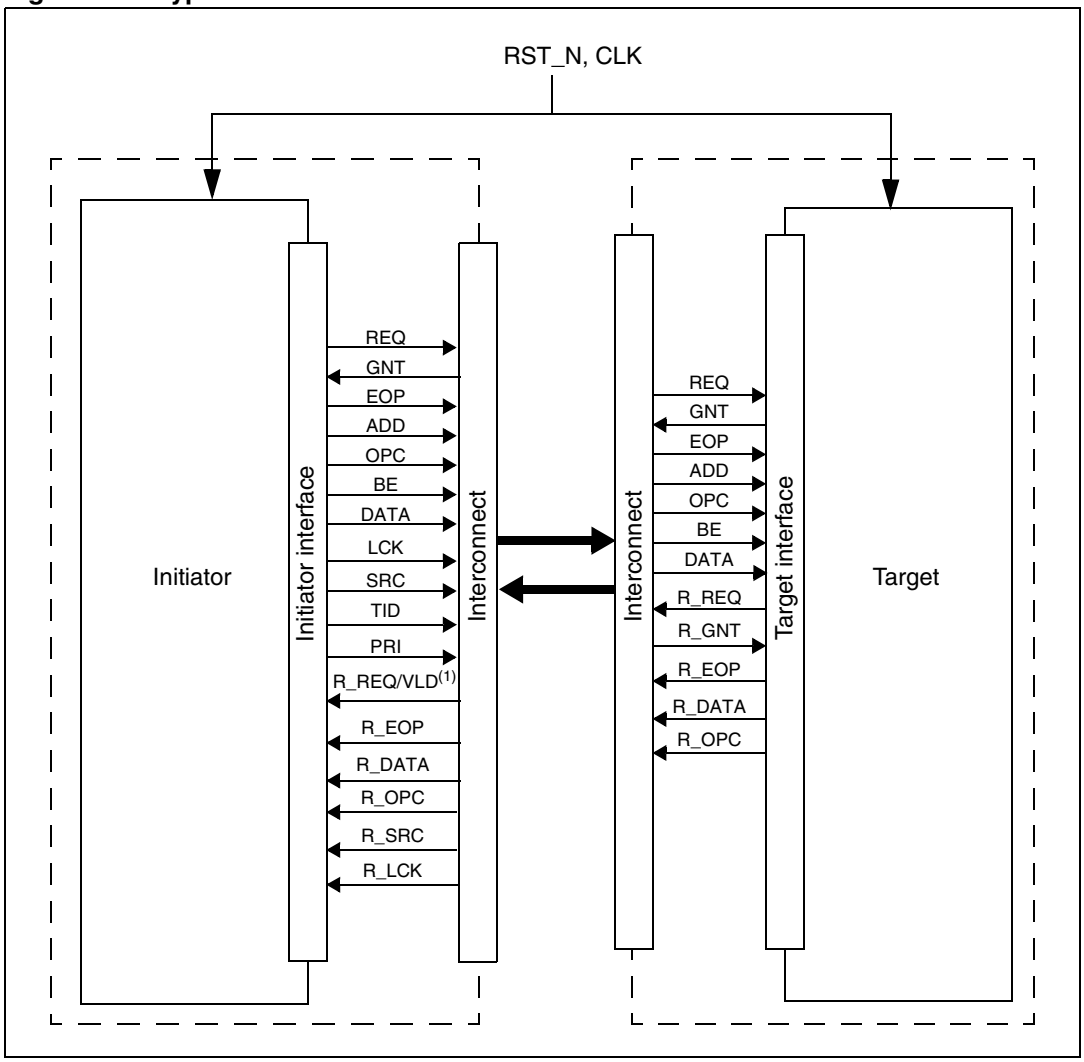
The type 2 or basic STBus interface increases the performance and functionality of the STBus ports. It supports all type 1 functionality and adds split transactions and the ability to support all transactions including compound operations, source labelling and some priority and transaction labelling/hint information.

It is targeted at devices which need high performance and pipelined operation but do not require the additional system efficiency associated with shaped request/response packets or the ability to re-order outstanding operations to improve performances.

Simple devices are required to support the following signals for a type 2 interface: REG, GNT, EOP, ADD, OPC, BE, DATA, R_OPC, R_EOP, R_DATA, however, certain operations also require support for LCK and the initiator may choose to implement additional optional functionality as shown by the signals described in the following section. The system also allows a robust error handling model to be implemented. While this is recommended, some systems may choose not to implement this functionality.

3.2.1 Type 2 interface definition

Figure 13. Type 2 interfaces



1. VLD is equivalent to R_REQ for initiators and can be assumed to be R_REQ in a system for which R_GNT is always '1'.

3.2.2 Type 2 signals and timings

The signals listed in *Table 10* are defined for type 2 initiators and targets. Signals which are mandatory for an interface are marked with 'M', signals which are optional and should only be implemented if required are marked with 'O'. Signals marked with '-' are not used.

Table 10. Type 2 signals and timings

Signal group	Full name	Signal name	Direction	Initiator		Target	
				Timing	Type	Timing	Type
Request flow control and atomicity and framing	request	REQ	init to target	early(1)	M	late	M
	grant	GNT	target to init	late	M	early	M
	end of packet(2)	EOP	init to target	early	O	late	M
	lock(3)	LCK	init to target	early	O	late	-
Request cell content	opcode	OPC[6:0]	init to target	early	M	late	M
		OPC[7](4)	init to target	early	O	late	M
	address	ADD[31:size(5)]	init to target	early	M	late	M
	byte enable(6)	BE[(2^{size}-1):0]	init to target	early	O	late	M
	data(7)	DATA[(8 * 2^{size}-1):0]	init to target	early	O	late	O
	source identity(8)	SRC[9:0]	init to target	early	O	late	-
	not end of message(9)	TID[4]	init to target	early	O	late	-
	write posting(10)	TID[5]	init to target	early	O	late	-
	store and forward(11)	TID[6]	init to target	early	O	late	-
	reserved(12)	TID[7]	init to target	early	O	late	-
	priority(13)	PRI[3:0]	init to target	early	O	late	-
Response flow control	response request	R_REQ	target to init	late	M	early	M
	response grant	R_GNT	init to target	late	-	early	M
	response end of packet(2)	R_EOP	target to init	late	O	early	M
	response lock(3)	R_LCK	target to init	late	O	early	-
Response cell content	response data(14)	R_DATA[(8 * 2^{size}-1):0]	target to init	late	O	early	O
	response source(15)	R_SRC[9:0]	target to init	late	O	early	-
	response not end of message(16)	R_TID[4]	target to init	late	O	early	-
	response opcode	R_OPC[0](17)	target to init	late	O	early	M
		R_OPC[7:1](18)	target to init	late	O	early	M

1. Early is defined as being in the first 20% of the clock cycle, late as being in the first 80% of the clock cycle and mid as being in the first 40% of the clock cycle.
2. Initiators may not implement it if they generate 1-cell packets only. Targets must implement it if for reusability reasons.
3. Initiators may not implement it if they don't generate chunks and RMW. Targets do not support it.
4. Initiators may not implement it since its meaning is not defined. Targets must implement if for reusability reasons.

5. Size defines the width of the interface, it may take a value between 0 and 4 and corresponds to interface widths of 1, 2, 4, 8, or 16 bytes (8, 16, 32, 64, or 128 bits).
6. Initiators may not implement it if they take care of entire words only. Targets must implement if for reusability reasons.
7. Not used if the initiator/target is read-only.
8. Initiators can allocate the low order bits if they contain multiple independent sources. If an initiator implements SRC[x:0] it is expected to also implement R_SRC[x:0]. Targets do not implement it.
9. Initiator can not use it if it doesn't generate messages. Targets do not implement it.
10. Initiator can not use it if it doesn't generate posted writes. Targets do not implement it.
11. Initiator can not use it if it doesn't require store and forward mechanism. Targets do not implement it.
12. Initiator can not use it since its meaning is not defined. Targets do not implement it.
13. Initiator can not use it if external priority based arbitration is not used inside the STBus. Targets do not implement it.
14. Not used if the initiator/target is write-only.
15. Initiators can allocate the low order bits if they contain multiple independent sources. Targets do not implement it.
16. Initiator can not use it if it doesn't need to have information about responses to messages. Targets do not implement it.
17. Initiators may not implement it if they neglect errors. Targets must implement if for reusability reasons.
18. Initiators may not implement it since its meaning is not defined. Targets must implement if for reusability reasons.

Note: The STBus building blocks always have all the signals at each interface; only external initiator and target IPs can use the optional signals. This will affect only the top-level STBus interconnect interface.

3.2.3 Type 2 enhancements/changes compared to type 1

With respect to the type 1 protocol, the type 2 protocol has the following main enhancements:

- support for split transactions
- support for pipeline

Support for split transactions means that a transaction is divided into two parts, the request part and the response part. The former starts with the couple REQ/GNT and ends with the signals REQ/GNT/EOP, the latter starts with the couple R_REQ/R_GNT and ends with the signals R_REQ/R_GNT/R_EOP.

Due to this feature, a type 2 initiator is free to do anything else when waiting for a response packet after the request packet has been completely transmitted.

Pipeline support is a consequence of split transaction support and means that an initiator can start a second transaction (in terms of a new request packet) without having received the answer to the first transaction. This can be true for more transactions and the maximum number of transactions in progress an initiator can have, without receiving a response, specifies the pipeline capability of the IP.

Independent of comparison with the type 1 protocol, type 2 protocol has two main distinctive features:

- symmetry
- order

Symmetry determines the equal shape between request and response packets, independent of the type of operation (for example, store or load). The request and the response packet are always composed of the same number of cells.

Order means, that if due to the pipeline, more transactions are generated, the sequence with which the responses are obtained must be exactly the same as the requests generated. To respect this protocol rule it may be required to add latency in the system, since it is forbidden for an initiator which has a pending transaction toward one target to start a new transaction toward a different target. This is because, if this was allowed and the second target was faster the first one, an order violation would occur.

To prevent this, a filtering mechanism is implemented within the STBus, forbidding an access toward a target if there is a pending access toward a different target. This filter introduces latency.

As far as the address is concerned, in case of multi-cell packets the address must be incremented cell by cell by the initiator, properly performing the address wrapping when required, in case of unaligned accesses.

3.2.4 Default grant definition

The grant signal (GNT) is defined as: a request has been recognized and the relative transaction has started. GNT can be generated in two different ways: causal and default (or early).

The causal grant is a GNT (normally low) asserted high as a reaction to a request. This is the behavior of the grant generated by a node toward an initiator when doing a request.

The default grant is a GNT (normally high) typically managed by the target interface of STBus converters and real targets such as type 2 and type 3 memory modules, used to tell the STBus they are available to get a new request and than manage a new transaction.

The default grant is normally de-asserted when the target (or the converter) has its internal FIFO full and no other transactions can be managed until at least one of the transactions in progress is completed by sending the relative response. The default grant is also called ‘early’ because it is generated by a state machine taking into account the status of the target (available/not available) and outputs directly from a flip-flop).

Note: The default grant is mandatory for all target modules. This allows the STBus node to ascertain whether a target is available or not to accept a new request. This avoids having to arbitrate among requests that cannot be serviced due to the unavailability of the addressed target.

The only case where the grant is generated “on request” in a combination of causal and default, is when it depends on a request arbitration process. This is because it occurs in the node in both the request and the response path.

3.2.5 Address usage

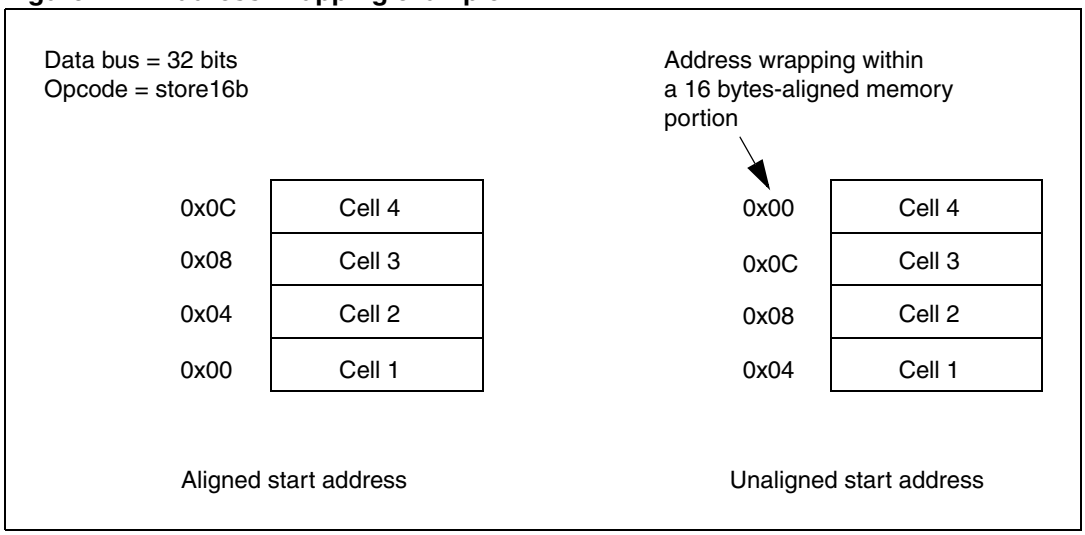
As far as the address of a packet is concerned, the following rule applies: the packet address must be opcode aligned, that is, it has to be a multiple of the number of bytes to be transferred within the packet.

This means that:

- for 1-byte operations valid addresses are 0x0, 0x1, 0x2, 0x3 and so on
- for 2-byte operations valid addresses are 0x0, 0x2, 0x4, 0x6 and so on
- for 4-byte operations valid addresses are 0x0, 0x4, 0x8, 0xC and so on
- for 8-byte operations valid addresses are 0x0, 0x8, 0x10, 0x18 and so on
- for 16-byte operations valid addresses are 0x0, 0x10, 0x20, 0x30 and so on
- for 32-byte operations valid addresses are 0x0, 0x20, 0x40, 0x60 and so on
- for 64-byte operations valid addresses are 0x0, 0x40, 0x80, 0xC0 and so on

If the start address of a packet is not transaction aligned, then a wrapping of the address has to be implemented, in order to keep the cells of the packet within the allowed memory portion aligned to the transaction. *Figure 14* shows the concept of address wrapping through an example.

Figure 14. Address wrapping example



Note: In type 1 and type 2 systems, the address wrapping for misaligned start addresses has to be performed by the initiators. In type 3 systems, the address wrapping can be performed by the targets (see Section 3.3.4: Address usage on page 52) as for store operations the address can be constant within a packet and for loads only one cell is generated.

3.2.6 Byte enables usage

As already described in *Section 3.1.4 on page 19*, byte enables (BE) is a signal whose size is equal to the number of bytes the data bus is composed of and may be defined in two ways:

- when the size of the operation (in bytes) is smaller than the data bus size (in bytes) the BE signal represents the lowest part of the address and must obey the addressing rule “the packet address must be opcode aligned”
- when the size of the operation is bigger than the data bus size, BE is simply a mask specifying which bytes of the overall word are affected by the operation

3.2.7 Opcode usage

The opcode encoding is spit into three sections.

- Bits [3:0] indicate the operation type: for example, load/store/cache operation.
- Bits [6:4] qualify the operation size: for example, the number of bytes the operation is on or the specific cache operation.
- Bit 7 is reserved and in multiplexed buses/systems is used to distinguish between request and response packets. The type 2 STBus is a non-multiplexed system and therefore bit 7 must be tied low.

For clarity, the possible opcodes are split into four operation types:

- *Primitive operations*
- *Compound operations*
- *Reserved operations*
- *Undefined operations*

Primitive operations

Primitive operations are the simplest operation types and are always built from a single request/response pair.

Table 11. Primitive operations

Packet type/operation	[7]	[6:4]	[3:0]
LD1: load byte	0	000	0001
LD2: load two bytes	0	001	0001
LD4: load four bytes	0	010	0001
LD8: load eight bytes	0	011	0001
LD16: load 16 bytes	0	100	0001
LD32: load 32 bytes	0	101	0001
LD64: load 64 bytes	0	110	0001
ST1: store byte	0	000	0010
ST2: store two bytes	0	001	0010
ST4: store four bytes	0	010	0010
ST8: store eight bytes	0	011	0010

Table 11. Primitive operations (continued)

Packet type/operation	[7]	[6:4]	[3:0]
ST16: store 16 bytes	0	100	0010
ST32: store 32 bytes	0	101	0010
ST64: store 64 bytes	0	110	0010
SWP4: swap 4 bytes	0	010	0101
SWP8: swap 8 bytes	0	011	0101

Compound operations

Compound operations are built from one or more primitives and may be constructed from multiple request/response pairs which may be linked by LCK, or have a causal relationship, the compound operations are defined in *Table 12*.

Table 12. Compound operations

Packet type/operation	[7]	[6:4]	[3:0]
PURGE: purge address	0	000	1000
FLUSH: flush address	0	001	1000
RMW4: READMODWRITE 4 bytes	0	010	0100
RMW8: READMODWRITE 8 bytes	0	011	0100
USER N: user defined	0	N	1111

Compound operations called chunks and messages can be built by linking together packets using the protocol signals lock (LCK) and not end of message (TID[4]), as described in *Section 3.2.9: Basic transactions description on page 36* and *Section 3.2.10: Chunk definition on page 40*.

Reserved operations

The operations specified in *Table 13* are defined, but not supported, within the current implementation of the protocols (for example, the group operations were used originally) or reserved for future implementations.

Table 13. Reserved operations

Packet type/operation	[7]	[6:4]	[3:0]
NOP: no operation	0	---	0000
EVENT	0	---	1110
LD128: load 128 bytes	0	111	0001
ST128: store 128 bytes	0	111	0010
RMW1: readmodwrite 1 byte	0	000	0100
RMW2: readmodwrite 2 bytes	0	001	0100
RMW16: readmodwrite 16 bytes	0	100	0100

Table 13. Reserved operations (continued)

Packet type/operation	[7]	[6:4]	[3:0]
SWP1: swap 1 byte	0	000	0101
SWP2: swap 2 bytes	0	001	0101
SWP16: swap 16 bytes	0	100	0101
LDG1: load group of bytes	0	000	1001
LDG2: load group of two byte quantities	0	001	1001
LDG4: load group of 4 byte quantities	0	010	1001
LDG8: load group of 8 byte quantities	0	011	1001
LDG16: load group of 16 byte quantities	0	100	1001
LDG32: load group of 32 byte quantities	0	101	1001
LDG64: load group of 64 byte quantities	0	110	1001
LDG128: load group of 128 byte quantities	0	111	1001
STG1: store group of bytes	0	000	1010
STG2: store group of two byte quantities	0	001	1010
STG4: store group of 4 byte quantities	0	010	1010
STG8: store group of 8 byte quantities	0	011	1010
STG16: store group of 16 byte quantities	0	100	1010
STG32: store group of 32 byte quantities	0	101	1010
STG64: store group of 64 byte quantities	0	110	1010
STG128: store group of 128 byte quantities	0	111	1010
successful response	1	---	---0
error response	1	---	---1

Undefined operations

The opcode values in *Table 14* are undefined.

Table 14. Undefined opcodes

Operation	[7]	[6:4]	[3:0]
undefined	0	---	0011
	0	---	0110
	0	---	0111
	0	---	1011
	0	---	1100
	0	---	1101

3.2.8 Response opcode usage

The target uses the response opcode signal to return information about the operation it has completed in terms of its status (success or failure) and other information characterizing it. This can include, for example, if it was a read or a write, how many bytes were addressed, where an error may have originated. It is used by converters to correctly generate the response traffic when crossing back different STBus domains.

The fields given below make up the response opcode signal.

- R_OPC[0]: The operation status (0 = success, 1 = failure), when R_OPC[0] = 0 then R_OPC[1] is redundant.
- R_OPC[1]: When R_OPC[1] = 0 the error has been originated by a target external to the interconnect. When R_OPC[1] = 1 the error has been originated by the interconnect itself as a result of an incorrect address, a security violation, or access to a target in power down mode.
- R_OPC[2]: This is reserved for future expansion in terms of error description.
- R_OPC[3]: The operation type (0 = write, 1 = read).
- R_OPC[6:4]: The operation size (copy of OPC[6:4]).
- R_OPC[7]: This is reserved, in multiplexed buses/systems R_OPC[7] is used to distinguish between request and response packets, since STBus type 2 protocol is a non-multiplexed system, bit 7 must be tied high.

3.2.9 Basic transactions description

A module communicates with the rest of the system using standard communication operations or transactions. These transactions define the operations which occur when a module exchanges information with the system. They are typically memory operations such as reads or writes of various sizes.

The transaction includes all the information required for the system to unambiguously determine the operation required and define the properties of the operation itself. These operations are defined in more detail in the following sections.

LOAD m bytes

Abbreviation	LD1, LD2, LD4, LD8, LD16, LD32, LD64
Definition	Read a single aligned word of m bytes from the target to the initiator. Valid sizes for m are defined to be 2^n where n is an integer in the range [0..6].
Qualifiers	ADD[31:n]: the address of the word to be accessed. BE[n-1:0]: the byte enables signal indicates which bytes within the word are significant. R_DATA [8 x 2^n-1:0]: data to be transferred, the significance of bytes within this field is inferred from the byte enables information. R_OPCODE: result of operation.
Comments	Load implements a read operation between in the initiator and the target.

STORE m bytes

Abbreviation	ST1, ST2, ST4, ST8, ST16, ST32, ST64
Definition	Write a single aligned word of m bytes from the initiator to the target, overwriting the location at that address with the data transferred. Valid sizes for m are defined to be 2^n where n is an integer in the range [0..6].
Qualifiers	ADD[31:n]: the address of the word to be accessed. BE[n-1:0]: the byte enables signal indicates which bytes within the word are significant. DATA[8 x 2^n-1:0]: data to be transferred.
Comments	Store implements a write operation between an initiator and a target. Some systems may use the TID[5] signal to implement write posting increasing the effective performance of the system.

READMODWRITE m bytes (Read Modify Write)

Abbreviation	RMW4, RMW8
Definition	Transfer the value of the aligned word of m bytes from the target to the initiator, leaving the target device 'locked' until a second transfer from the initiator to the target completes, replacing the information held at the specified address in the target device. Valid sizes for m are defined to be 2^n where n is an integer in the range [2..3].
Qualifiers	ADD[31:n]: the address of the word to be accessed. BE[n-1:0]: The byte enables signal indicates which bytes within the word are significant. DATA[8 x 2^m-1:0]: data to be transferred from the initiator to the target. R_DATA[8 x 2^n-1:0]: data to be transferred from the target to the initiator.
Comments	READMODWRITE is an atomic operation in which the data manipulation occurs locally to the initiator. The system ensures that the target is unavailable to the rest of the system during this period impacting performance in systems with multiple masters, but which requires minimal support by the initiator. READMODWRITE has been introduced to have a 1 to 1 mapping to the test and set assembler instruction. This simplifies module implementation since both the initiator and the target may treat the READMODWRITE as a read operation followed by a write to the same address. This guarantees that once the target has been locked, it will be unlocked at the end of the transaction. Allowing the read and the write to be performed at different addresses implies an access risk to different targets, leaving the read locked indefinitely. It is possible to perform an atomic operation consisting of a read followed by a write at different addresses (but within the same target) generating a chunk (see *Section 3.3.9: Basic transaction description on page 54*).

SWAP m bytes

Abbreviation	SWP4, SWP8
Definition	Exchange the value of the single aligned word of m bytes from the initiator with the data held in the specified target location, returning the original target data to the initiator. Valid sizes for m are defined to be 2^n where n is an integer in the range [2..3].
Qualifiers	ADD[31:n]: the address of the word to be accessed. BE[n-1:0]: the byte enables signal indicates which bytes with the word are significant. DATA[8 x 2^n-1:0]: data to be transferred from the initiator to the target. R_DATA[8 x 2^n-1:0]: data to be transferred from the target to initiator.
Comments	SWAP is an atomic operation in which the data manipulation occurs locally to the target, replacing the current value with a new value and returning the old value to the initiator. Both the initiator and the target remain accessible to the system during this operation. The implementation of the target is a little more complex due to the requirement for local storage, but performance is higher due to no 'dead' periods existing during which the target is unavailable.

FLUSH address

Abbreviation	FLUSH
Definition	Returns a response when any copies of the data associated with the physical address (which may be held by the target module) are coherent with the actual data held at the physical address. The target device may retain a copy of that data.
Qualifiers	ADD[31:0]: the address of the target module potentially holding a copy of the word to be flushed DATA[31:0]: physical address of the word to be flushed, organized MSB to LSB, bit 0 containing the LSB
Comments	FLUSH is an operation used to ensure coherence of the main memory whilst allowing local copies associated with a target to remain coherent. Typically this is used in conjunction with a coherent LDm operation as follows: ● initiator A wants to read a location at address ADD from target C (a memory) but knows target B (a cache) may have a local copy ● initiator A passes FLUSH ADD to target B ● if target B has no copy it passes a response to initiator A ● if target B has incoherent copy, it performs STm ADD to target C and passes a response to A on completion

- initiator A performs LD ADD to target C
- initiator A receives a coherent copy from target C whilst the local copy in target B also remains coherent

PURGE address

Abbreviation	PURGE
Definition	Returns a response when any copies of data associated with the physical address (which may be held by the target module) are coherent with the actual data held at the physical address, and removes any copies of the data held within the target module.
Qualifiers	ADD[31:0]: the address of the target module potentially holding a copy of the word to be purged. DATA[31:0]: physical address of the word to be purged, organized MSB to LSB, bit 0 containing the LSB.
Comments	This operation is used to ensure coherence of main memory whilst ensuring stale local copies are destroyed. Typically this is used in conjunction with an STm operation as follows: ● initiator A wants to write to a location at address ADD in target C (a memory) but knows target B (a cache) may have a local copy ● initiator A passes PURGE ADD to target B ● if target B has no copy it passes a response to initiator A ● if target B has incoherent copy, it performs STm ADD to target C, destroys it's local copy and passes a response to A on completion ● initiator A performs STm ADD to target C

User operation type m

Abbreviation	USER 0:7
Definition	These opcodes are reserved for special user functions and may be used to implement functions or communications which are required in a particular system. The target device (if it supports that operation in this system) should return a response to the initiator as defined by the user after completing the operation.
Comments	The target may qualify the user defined operation by examining the source of the operation and using this information to distinguish between user operations from different initiators. Up to eight user operations may be defined for a specific system.

Unsupported operations

Definition	Return a response on reception of the operation.
Comments	The target device should always return an error when accessed with an operation which is reserved. This allows the operation set to be extended for future systems without making existing IPs redundant.

3.2.10 Chunk definition

A chunk is a collection of packets linked by the lock (LCK) signal. The following rules characterize chunks.

- A chunk is a set of packets linked together by the LCK signal.
- LCK is high for all the packets of the chunk except for the last packet.
- LCK has the following meaning: if an initiator generates a packet with the LCK set to '1', the accessed target can only be accessed by that initiator until it sends a packet with the LCK set to '0'. STBus must guarantee this.
- The initiator may assume that for a locked sequence of operations:
  - the operations are maintained by the system as an atomic group
  - the order of the response packets is the same as the order of request packets for that sequence

  Implementation is achieved by enforcing the properties described below.
  - If LCK is asserted for packet n from initiator A to target B, then that target may only be accessed by initiator A until it receives a subsequent packet n + 1 from initiator A with LCK de-asserted. The order of packets n and n + 1 is always maintained.
  - If R_LCK is asserted for packet m from target A to initiator B, then that initiator may only be accessed by target A until it receives a subsequent packet m + 1 from target A with R_LCK de-asserted. The order of response packet m and m + 1 is always maintained.

  The target module ensures that the generated response packets have the same order as the request packets presented.
- All operations within a chunk must be from the same initiator (the SRC must be constant).
- All operations within a chunk should be within an aligned 256-byte address space. Operations must be transaction aligned and at most they can be 256 bytes aligned.
- For a type 3 system, all operations within a chunk must be uniquely labelled using TID[3:0][(a)].
- All operations within a chunk should be of the same type (that is, the opcode should be constant) in order to guarantee optimum performances from the system. Chunks built of packets with different opcodes are supported.
- An initiator should not slow send a chunk (that is, there should be no cycle gaps between operations within a chunk).
- The size of a chunk can be programmable inside an IP and it can range from 1 byte to 256 bytes.
- Within all the packets of a chunk, the same target must always be addressed.

a. Recommendation.

- Order between the response packets must be guaranteed for a correct operation of the IP generating the chunk. The target must guarantee this order.
- The interconnect is not able to recover if the IP, for any reason, stops the transfer forgetting the lck high. In this case, the target addressed during the chunk transmission would remain blocked by that initiator and no other initiators can access it. This could cause the system to stall.

3.2.11 Message definition

A message is a set of chunks linked together by the “not end of message” (TID[4]) signal. Chunks that build a message can be composed of a single packet. It is therefore possible to have messages composed of packets linked together by the TID[4] signal.

The concept of a message has been introduced in order to optimize access to memory devices such as DRAM, SDRAM and DDR SDRAM memory.

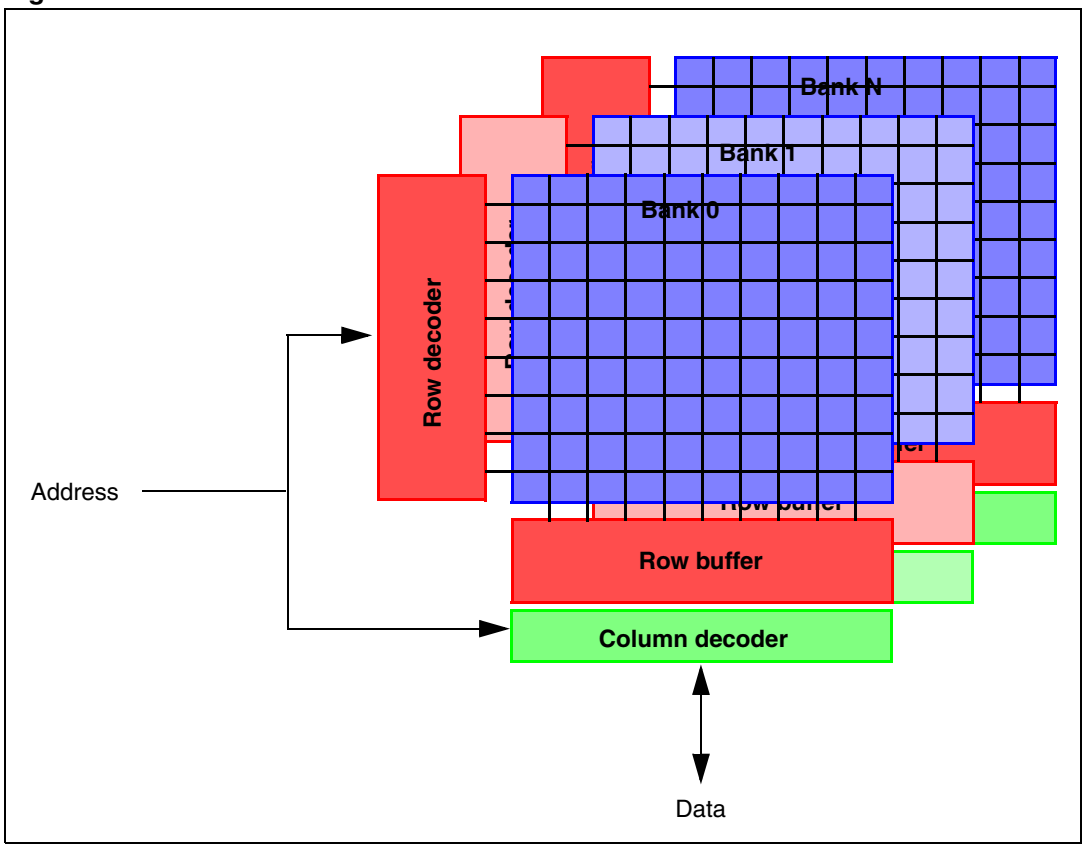
Modern computer systems have become increasingly limited by memory performance; while processor performance increases at a rate of 60% per year, the bandwidth of a memory chip (DRAM) increases by only 10% per year.

To maximize memory bandwidth, modern DRAM components allow pipelining of memory accesses by means of a three dimensional structure, based on the concepts of bank, row (or page) and column (refer to *Figure 15*).

Banks can be accessed independently and the most recently accessed pages are cached. As a result, sequential accesses to different pages within one bank have high latency and cannot be pipelined, while accesses to different banks or different words within a single page have low latency and can be pipelined.

This three dimensional structure makes it advantageous to reorder memory operations in order to exploit the non-uniform access times of the DRAM.

Figure 15. DRAM basic structure



For STBus, a page is a portion of memory n bytes wide and aligned on an n-byte boundary, with n being 64, 128, 256, 512, 1K, 2K, 4K, 8K or 16K.

As previously stated, page data is only accessible if that page has previously been opened. Each page opening incurs an overhead of several clock cycles before the data can be accessed.

In order to minimize the number of page openings, the interconnect has to arbitrate on a larger amount of data than that contained within a single packet. Since a message groups together operations aimed at the same page of memory, an arbitration scheme based on messages rather than on packets is suitable to guarantee continuous DRAM/SDRAM/DDR accesses minimizing the number of page misses and increasing the memory efficiency.

It is important to outline the difference between physical page and logical page.

- The logical page is a portion of the physical page and different logical pages are usually mapped into different memory banks.
- A physical page is in the same memory bank.

The physical page size is equivalent to, a power of 2 multiple of the logical page size, in the same memory bank.

In an example system used for video processing: when generating addresses in raster mode, logical pages are accessed, but with DDR memory accesses this results in jumping from one part of a physical page to another, the same is usually true for banks.

Message characteristics

The following set of rules characterize a message.

- The TID[4] is high for all the chunks of the message except the last one.
- The meaning of the TID[4] is as follows: if an initiator is sending a packet marked by the TID[4] set to '1', it means the next packet sent by that initiator may be linked to the current one. The way in which this is handled depends on the particular STBus system implementation.
- It is strongly recommended that a message is no more than 256 bytes wide (the size of a memory page). All the packets of a message should address the same memory page within the same target. The STBus and the accessed target must be able to operate correctly even if this recommendation is not respected.
- An IP must terminate a message.
- If for any reason (such as a fault) the IP doesn't terminate a message, the STBus and the target must be able to manage this occurrence without stalling the system.
- An initiator should not slow send messages (that is, there should be no cycle gaps between operations within a message).

3.2.12 Examples

Single cell operations

Figure 16. Single cell transfer from an initiator

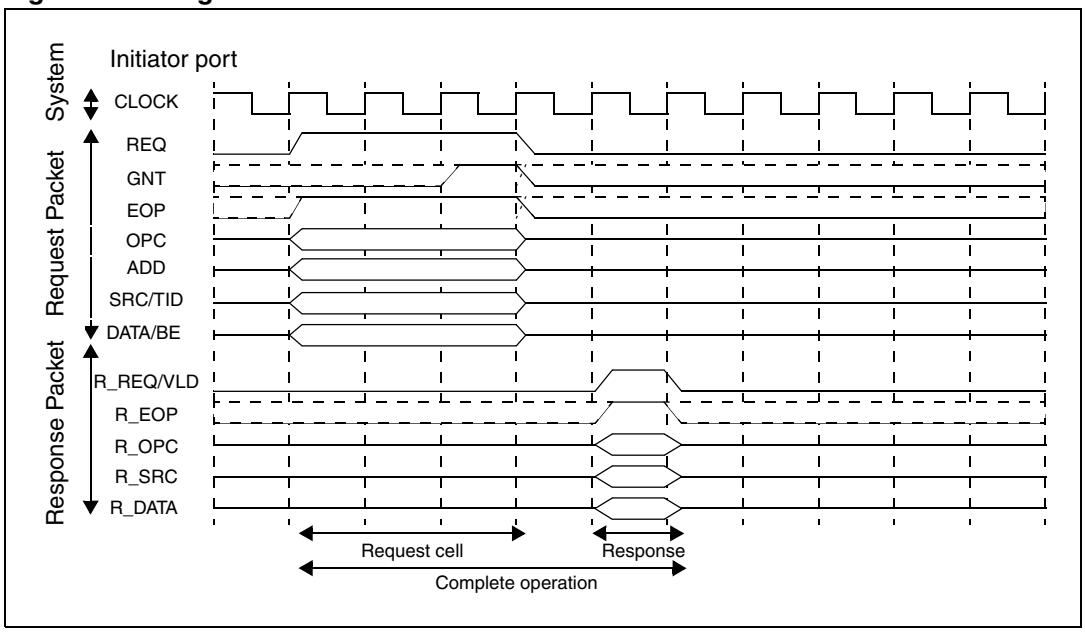
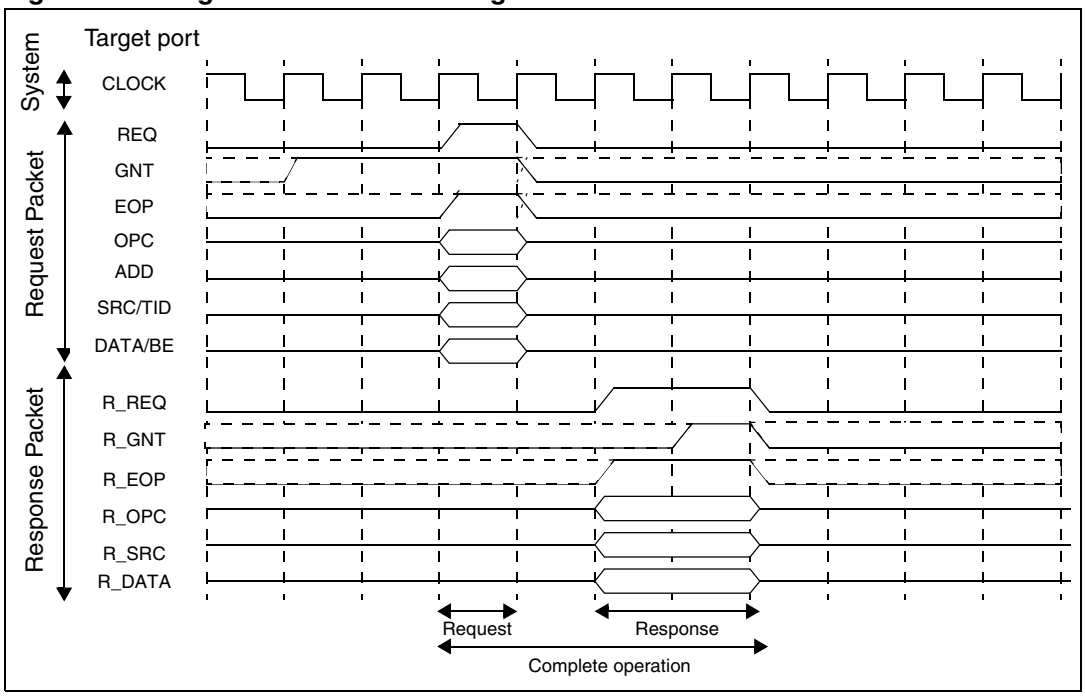


Figure 17. Single cell transfer to a target



Two cells operations

Figure 18. Two cell transfer from an initiator

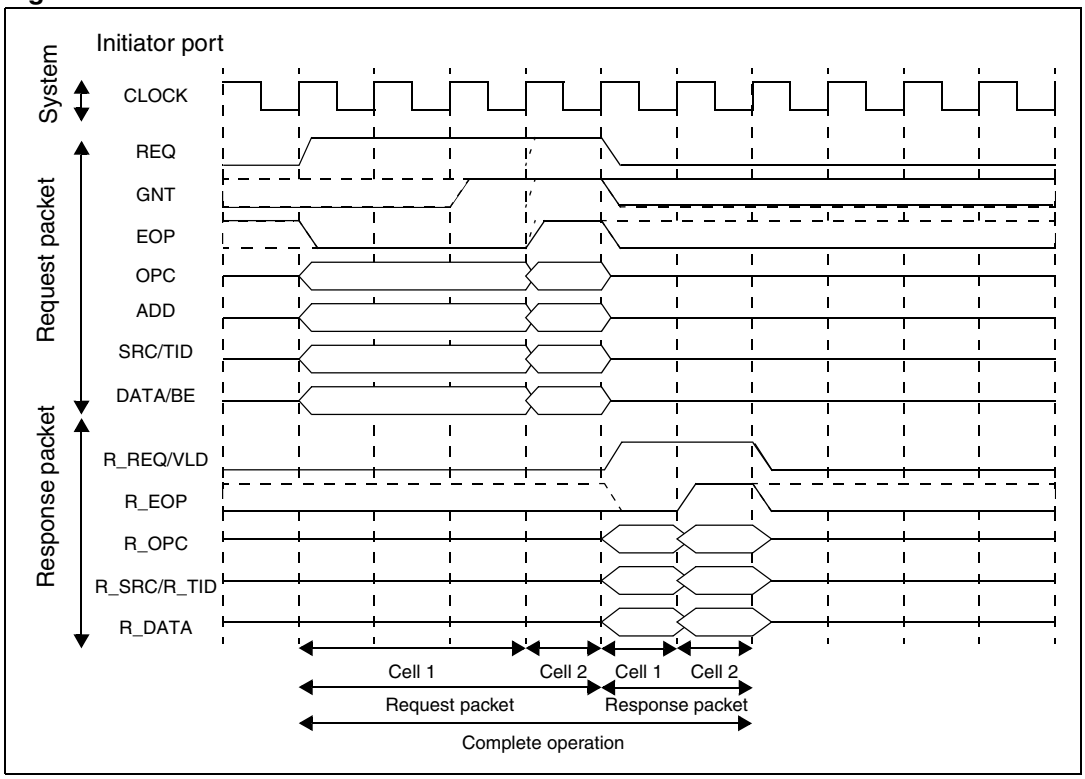
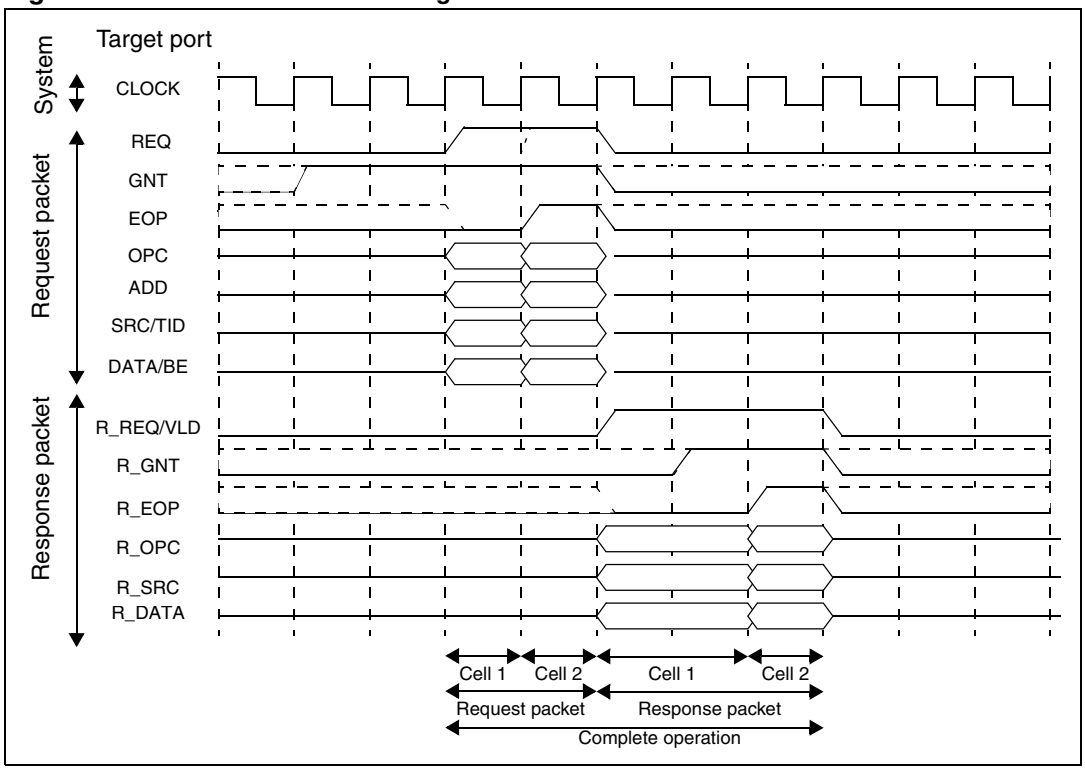


Figure 19. Two cell transfer to a target



Four cells operations

Figure 20. Four cell transfer from an initiator

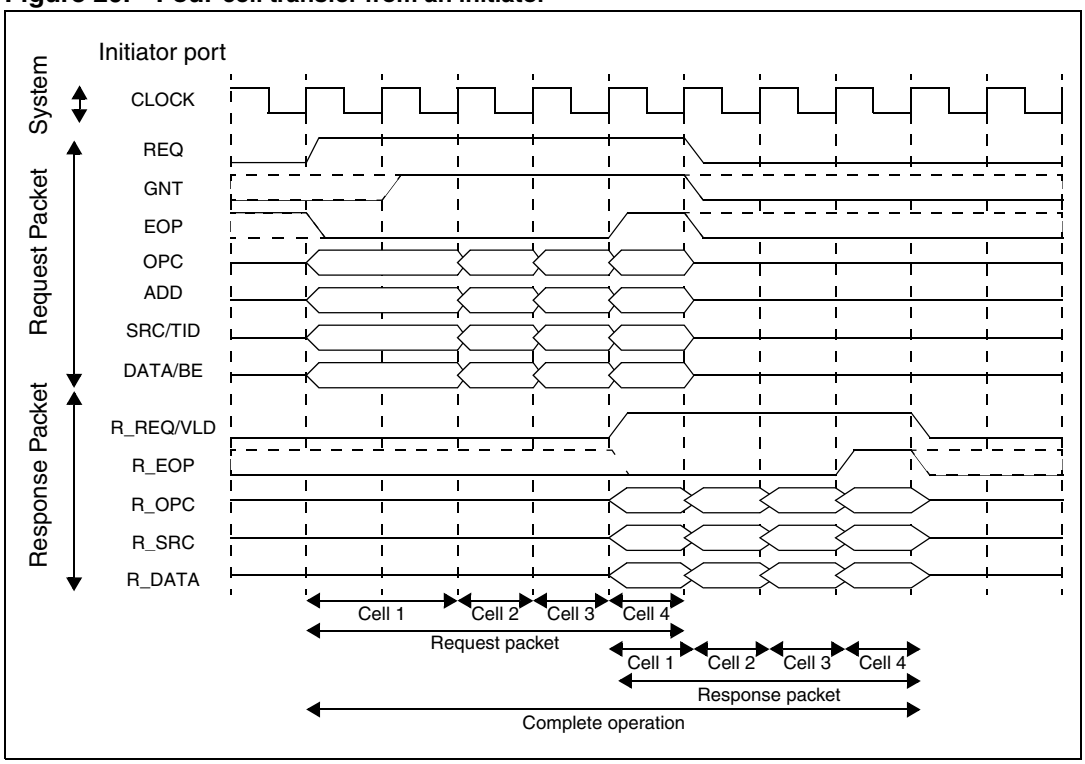
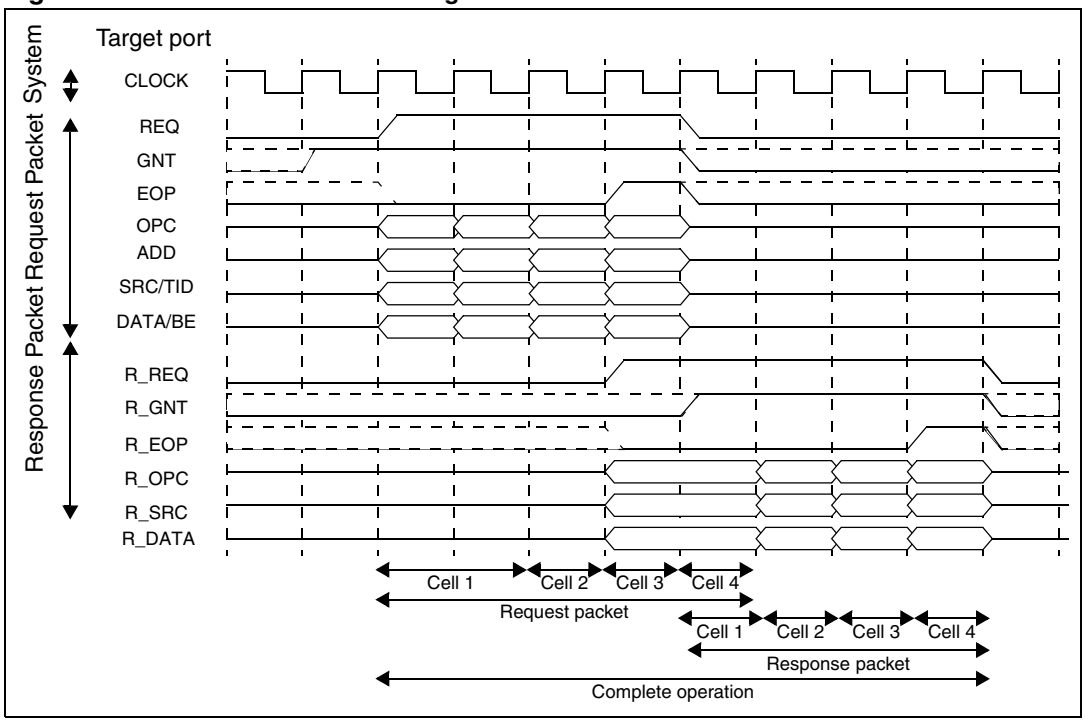


Figure 21. Four cell transfer to a target



Single cell RMW

Figure 22. Single cell RMW transaction from an initiator

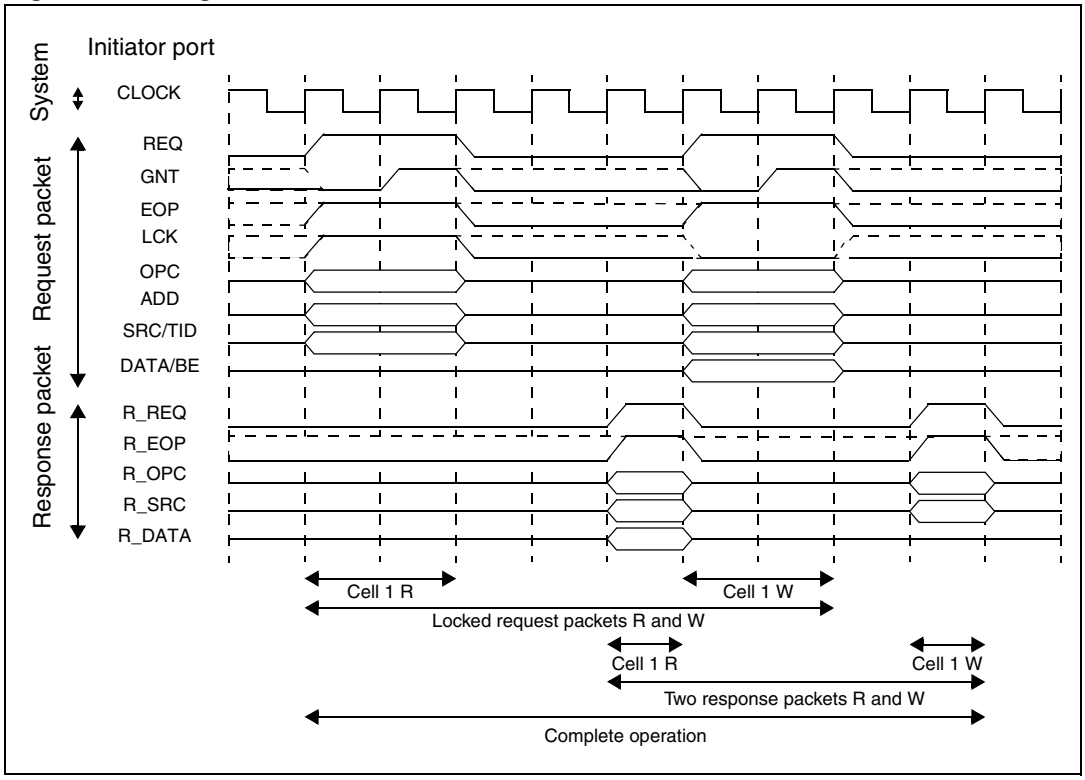
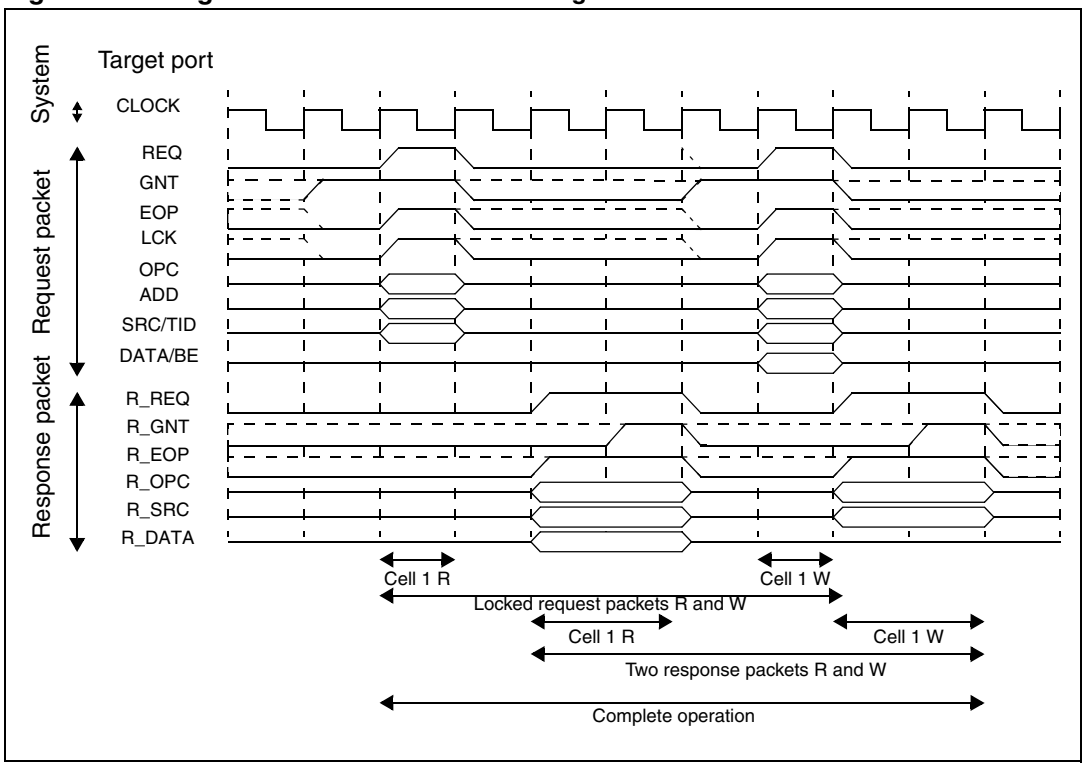


Figure 23. Single cell RMW transaction to a target



Two cells RMW

Figure 24. Two cells RMW transaction from initiator

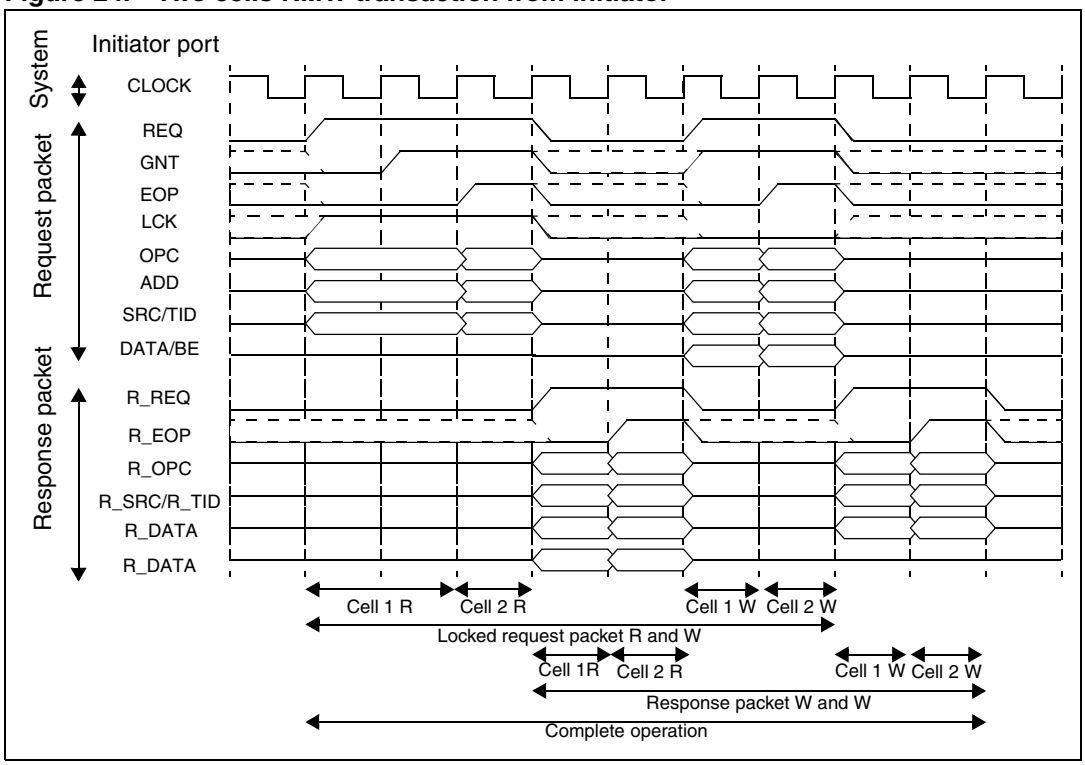
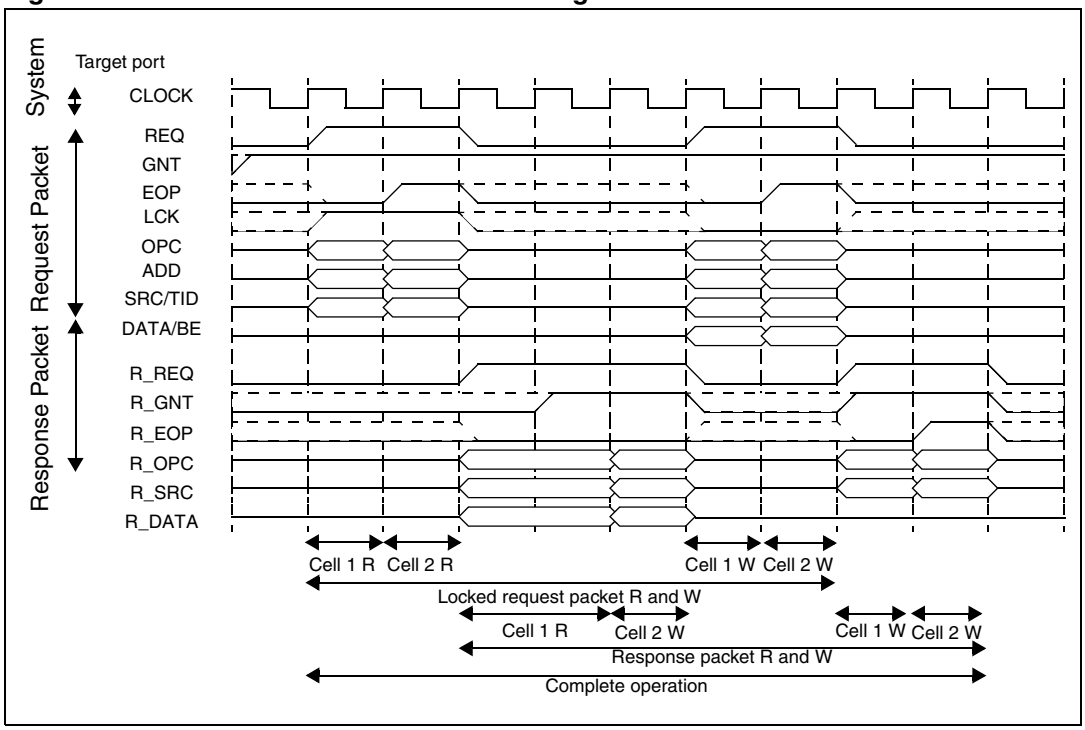


Figure 25. Two cells RMW transaction to target



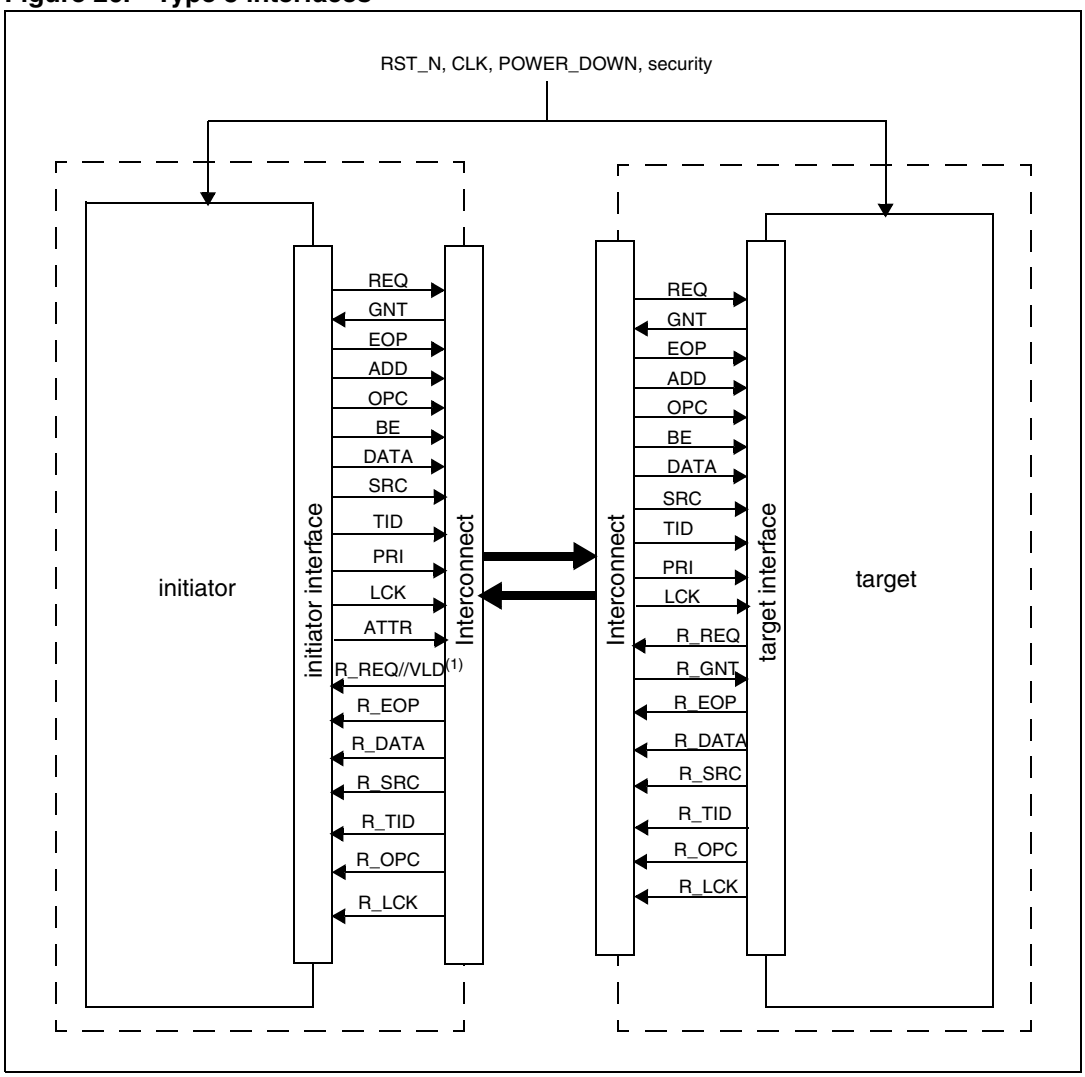
3.3 Type 3 protocol (advanced)

The type 3 or advanced STBus interface increases the performance and functionality of the STBus port. As the type 2 protocol, it supports all type 1 functionality and adds split transactions and the ability to support all transactions including compound operations, source labelling and some priority and transaction labelling/hint information.

It is targeted at devices which need high performance and pipelined operations and, with respect to type 2 protocol, gives the additional feature of shaped request/response packets and the ability to re-order outstanding operations to improve performance.

3.3.1 Type 3 interfaces definition

Figure 26. Type 3 interfaces



1. VLD is equivalent to R_REQ for initiators and can be assumed to be R_REQ in a system for which R_GNT is always '1'.

3.3.2 Type 3 signals and timings

The signals listed in *Table 15* are defined for type 3 protocol. Signals which are mandatory for the interface are marked with 'M', signals which are optional and should only be implemented if required are marked with 'O'. Signals marked with '-' are not used.

Table 15. Type 3 signals and timings

Signal group	Full name	Signal name	Direction	Initiator		Target	
				Timing	Type	Timing	Type
Request flow control and atomicity and framing	request	REQ	init to target	early(1)	M	late	M
	grant	GNT	target to init	late	M	early	M
	end of packet(2)	EOP	init to target	early	O	late	M
	lock(3)	LCK	init to target	early	O	late	M
Request cell content	opcode	OPC[6:0]	init to target	early	M	late	M
		OPC[7](4)	init to target	early	O	late	M
	address	ADD[31:size(5)]	init to target	early	M	late	M
	byte enable(6)	BE[(2^{size}-1):0]	init to target	early	O	late	M
	data(7)	DATA[(8 * 2^{size}-1):0]	init to target	early	O	late	O
	source identity(8)	SRC[9:0]	init to target	early	O	late	M
	transaction identity	TID[3:0]	init to target	early	M	late	M
	not end of message(9)	TID[4]	init to target	early	O	late	M
	write posting(10)	TID[5]	init to target	early	O	late	M
	store and forward(11)	TID[6]	init to target	early	O	late	M
	reserved(12)	TID[7]	init to target	early	O	late	M
	priority(13)	PRI[3:0]	init to target	early	O	-	-
	protection attribute(14)	ATTR[3:0]	init to target	early	O	late	O
	cacheability attribute(15)	ATTR[7:4]	init to target	early	O	late	O
	addressing policy attribute(16)	ATTR[9:8]	init to target	early	O	late	O
	user defined attribute(17)	ATTR[15:10]	init to target	early	O	late	O
Response flow control	response request	R_REQ	target to init	late	M	early	M
	response grant	R_GNT	init to target	-	-	early	M
	response end of packet(2)	R_EOP	target to init	late	O	early	M
	response lock(3)	R_LCK	target to init	late	O	early	M

Table 15. Type 3 signals and timings (continued)

Signal group	Full name	Signal name	Direction	Initiator		Target	
				Timing	Type	Timing	Type
response cell content	response data(18)	R_DATA[(8 * 2^{size}-1):0]	target to init	late	O	early	O
	response opcode	R_OPC[7:0]	target to init	late	O	early	M
	response source(19)	R_SRC[9:0]	target to init	late	O	early	M
	response transaction identity	R_TID[3:0]	target to init	late	M	early	M
	response not end of message(20)	R_TID[4]	target to init	late	O	early	M
	response write posting(20)	R_TID[5]	target to init	late	O	early	M
	response store and forward(11)	R_TID[6]	target to init	late	O	early	M
	reserved(12)	R_TID[7]	target to init	late	O	early	M

1. Early is defined as being in the first 20% of the clock cycle, late as being in the first 80% of the clock cycle and mid as being in the first 40% of the clock cycle.
2. Initiators can not implement it if they generate 1-cell packets only. Targets implement it for reusability reasons.
3. Initiators can not implement it if they don't generate chunks and RMW. Targets implement it for reusability reasons.
4. Initiators can not implement it since its meaning is not defined. Targets implement it for reusability reasons.
5. Size defines the width of the interface, it may take a value between 0 and 4 and corresponds to interface widths of 1, 2, 4, 8, or 16 bytes (8, 16, 32, 64, or 128 bits).
6. Initiators can not implement it if they take care of entire words only.
7. Not used if the initiator/target is read-only. Targets implement it for reusability reasons.
8. Initiators allocate the low order bits if they contain multiple independent sources. If an initiator implements SRC[x:0] it is expected to also implement R_SRC[x:0]. Targets implement it to generate a copy when responding.
9. Initiators can not use it if it doesn't generate messages. Targets implement it for reusability reasons.
10. Initiators can not use it if it doesn't generate posted writes. Targets implement it for reusability reasons.
11. Initiators can not use it if they don't require store and forward. Targets implement it for reusability reasons.
12. Initiators can not use it since its meaning is not defined. Targets implement it for reusability reasons.
13. Initiators can not use it if external priority based arbitration is not implemented in STBus. Targets do not implement it
14. The ATTR[3:0] bits are optionally implemented to provide protection attribute. Targets do not support them.
15. The ATTR[7:4] bits are optionally implemented to provide cacheability attribute. Targets do not support them.
16. The ATTR[9:8] bits are optionally implemented to provide addressing policy attribute. Targets do not support them.
17. The ATTR[15:10] bits are optionally implemented to provide user defined attributes. Targets do not support them.
18. Not used if the initiator/target is write-only.
19. Initiators can allocate the low order bits if they contain multiple independent sources. Targets implement it for reusability reasons.
20. Initiators do not use it. Targets implement it for reusability reasons.

Note: The STBus interconnect top level and building blocks always have all the signals at each interface; only external initiator and target IPs can use the optional signals.

3.3.3 Type 3 enhancements/changes compared to type 2

The type 3 protocol has two main features differentiating it from the type 2 protocol, these features are the shaped packets and the out of order management capability.

Shaped packets

Shaped packets allow optimum bandwidth allocation, since the minimum number of clock cycles to carry out a transaction is used.

A store operation consists of a request packet (composed of a number of cells depending on the amount of data to store) and a response packet (composed of only one cell) indicating that the whole store operation has been completed.

A load operation consists of a request packet composed of only one cell, specifying the amount of data to read from memory (by the opcode) and the first address from which data must be read, and a response packet composed of a number of cells depending on the amount of data to read.

Due to this asymmetry between request and response packets, it's clear that the bandwidth allocation in a type 3 system is optimized compared to a type 2 system. In a type 2 system, the number of cycles required to carry out a transaction does not depend on the type of the operation.

Out of order

The type 3 protocol allows initiators to get responses in a different order to the sequence of requests.

This allows a gain in terms of latency, because the filtering mechanism required in type 2 nodes (to prevent an initiator having a transaction in progress to access a different target) is not required.

To manage the out of order type 3 IPs use the transaction identifier signal TID[3:0]. This means that any IP can have up to 16 different transactions in progress.

3.3.4 Address usage

Asymmetry within the type 3 protocol, leads to a single-cell load request packet. Therefore, type 3 targets must be able to internally increase the address and (if required for misaligned accesses) to wrap the address, in order to deliver all the data required with the request packet generated from the opcode. This mechanism allows the address in the different cells of a store request packet to be kept constant. This is the opposite of what happens in type 2 systems where it is up to the initiator to correctly increase the address cell by cell. performing the proper wrapping when there are misaligned accesses.

3.3.5 Byte enables usage

As in type 1 and type 2 protocols, byte enables (BE) is a signal whose size is equal to the number of bytes the data bus is composed of, and may be defined in two ways:

- when the size of the operation (in bytes) is smaller than the data bus size (in bytes) the BE signal represents the lowest part of the address and must obey the addressing rule "the packet address must be opcode aligned"
- when the size of the operation is bigger than the data bus size, the BE is simply a mask specifying which bytes of the overall word are affected by the operation

In type 3 systems, load operations are characterized by only one request cell. The BE signal associated to this request cell is applied only to the first cell of the load operation by the target. For the other cells to be read, the target assumes that all the bits of the BE signal are set to '1'.

3.3.6 Opcode usage

The usage of the opcode in type 3 is basically the same as for type 2 (refer to *Section 3.2.7: Opcode usage on page 32*). The same operations are defined, and the same encoding of the opcode is used.

3.3.7 Response opcode usage

The usage of the response opcode in type 3 is exactly the same as for type 2 (refer to *Section 3.2.8: Response opcode usage on page 35*). The same fields are defined, and the same meaning is attributed to them.

3.3.8 Attribute signal usage

This signal is used by the initiator to label a transaction with an additional attribute field, in order to transmit the following information:

- protection (PROT[3:0])
- cacheability (CACHE[3:0])
- addressing policy type (PTYPE[1:0])

In addition, a set of six optional bits is reserved for user-defined applications.

The protection attribute signal is used to propagate to targets, information about the privilege level (ATTR[0]) and data/instruction fetch (ATTR[2]). The security bit (ATTR[1]) is instead used by the STBus to apply security instructions.

The exclusive lock (ATTR[3]) is used to implement semaphores: if a target is marked with the ELCK signal by an initiator and another initiator tries to access it, the target responds with an error. The ELCK signal is transparent to the STBus interconnect.

The cache attribute (ATTR[7:4]) signal is transparent to the STBus interconnect and is used simply to propagate cacheability information to targets.

The explicit addressing policy type or PTYPE (ATTR[9:8]) signal allows STBus type 3 initiators to generate asymmetric load/store commands to AMBA AHB/APB bridges. This offers a gain in terms of bandwidth usage and the bridge's FIFO size configuration.

Note: AHB/APB devices usually have only one port to access registers and FIFOs and they do not contain appropriate logic to uncompact/compact data from/to a bus with higher width (for example, STBus native targets). The appropriate size of transfer must be handled by initiators and transmitted through the interconnect.

This signal guarantees efficient peripheral FIFO static accesses but it also keeps the performance of AHB memory controller devices that exploit incremental and wrapped AHB transfers. AHB/APB bridges have to support the PTYPE signal otherwise the asymmetry cannot be exploited and transfers similar to those used by type 2 must be generated by initiators.

In order to reconstruct the AHB/APB transfers correctly, the three following fields must be coherent: PTYPE, BE, OPC[6:4].

- The PTYPE signal indicates what kind of transfer must be generated: single (static), incremental, wrapped.
- The BE signal must be coherent with the position of valid bytes (LSB address). For incremental or wrapped transfers it is mandatory to always have the total number of asserted be bits coherent with the AHB/APB transfer size, as it is the only way to reconstruct this information. For "static burst", the number of asserted BE bits must be enough to allow correct size conversion (done by the AXI/AHB bridge).
- OPCn indicates the number of response cells (n) that must be generated by the AHB/APB bridges. It is always equal to: n*(bus width/8) for 64-bit initiators.

The optional six attribute bits (ATTR[15:10]) can be used to implement operations specific to the system, issued by initiators and executed by targets. They are transparent to the STBus interconnect, whose task is simply to propagate these commands from the originating initiator to the selected target.

The attribute signal must be kept constant for the duration of the whole transaction (message, chunk or packet).

3.3.9 Basic transaction description

The transactions in type 3 are described in exactly the same way as for type 2 (refer to *Section 3.2.9: Basic transactions description on page 36*), taking into account the different shape of the request and the response packets.

3.3.10 Chunks and messages description

In type 3 the chunks and the messages are implemented and managed in exactly the same way as in type 2 (refer to *Section 3.2.10: Chunk definition on page 40* and *Section 3.2.11: Message definition on page 41*).

3.3.11 Examples

Single cell operations

The shape of single cell transactions in type 3 is exactly the same as for type 2, for both store and load (refer to *Section 3.2.12: Examples on page 43*).

Two cells response

Figure 27. Two cells response at Initiator interface

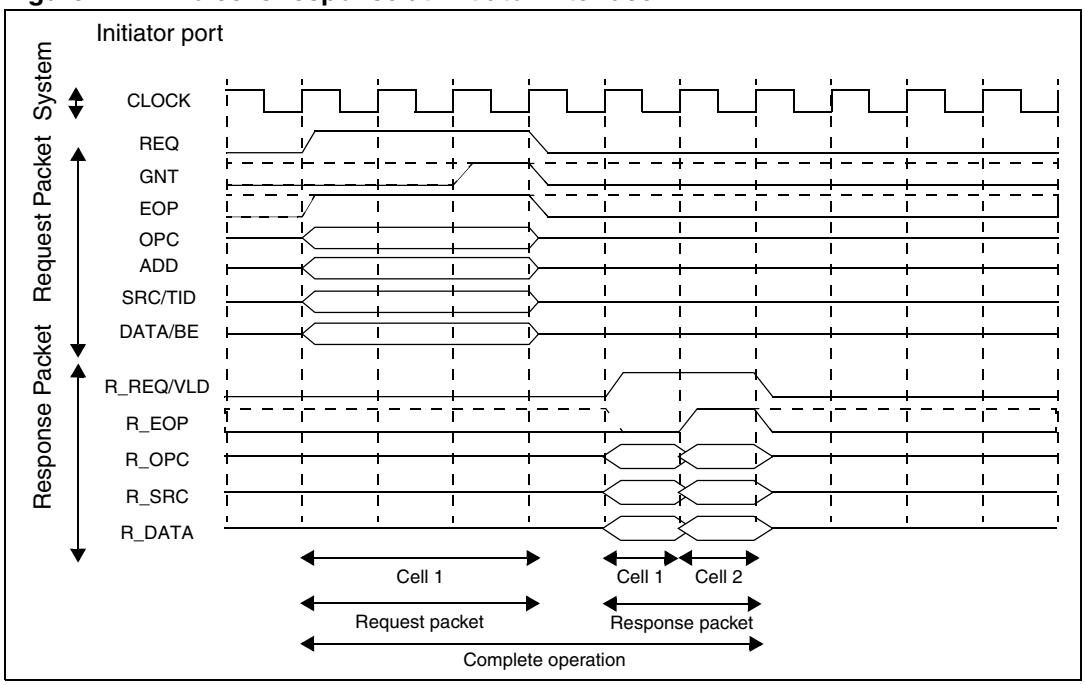
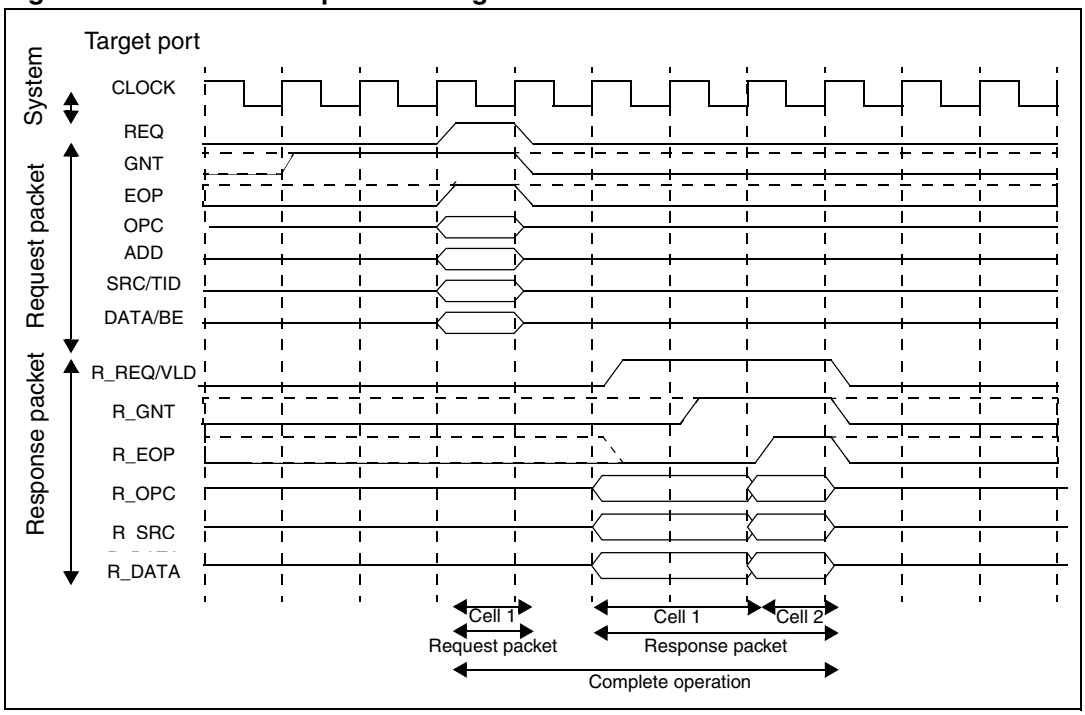


Figure 28. Two cells response at target interface



Four cells response

Figure 29. Four cells response at initiator interface

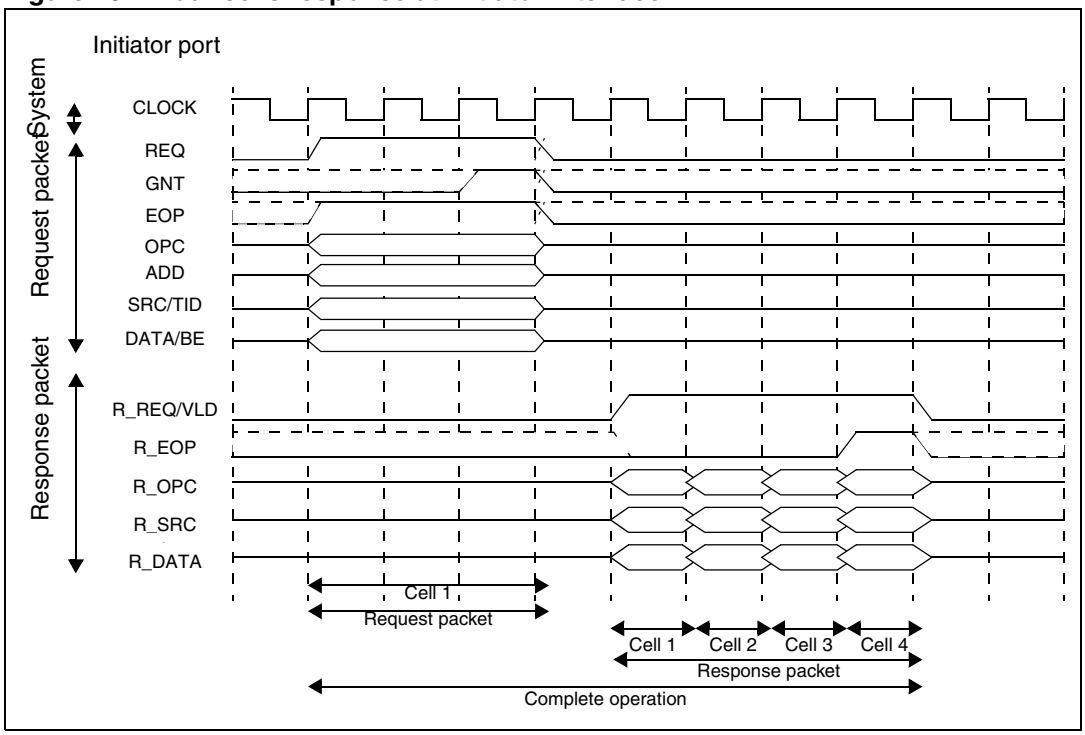
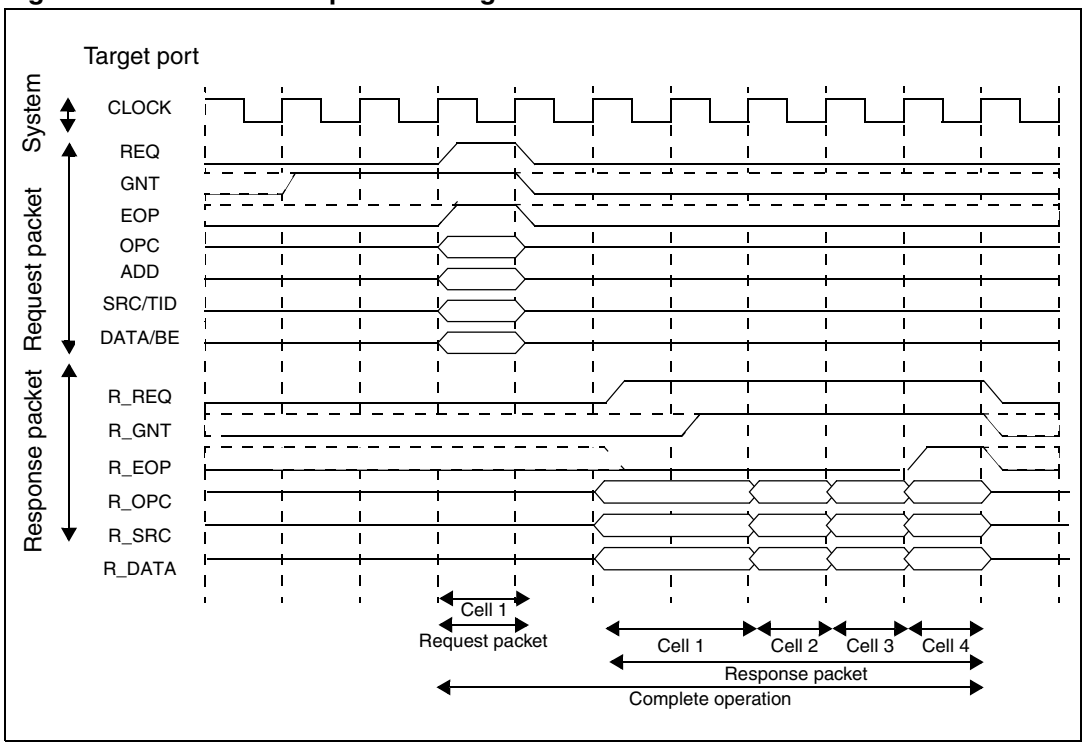


Figure 30. Four cells response at target interface



Two cells request

Figure 31. Two cells request at initiator interface

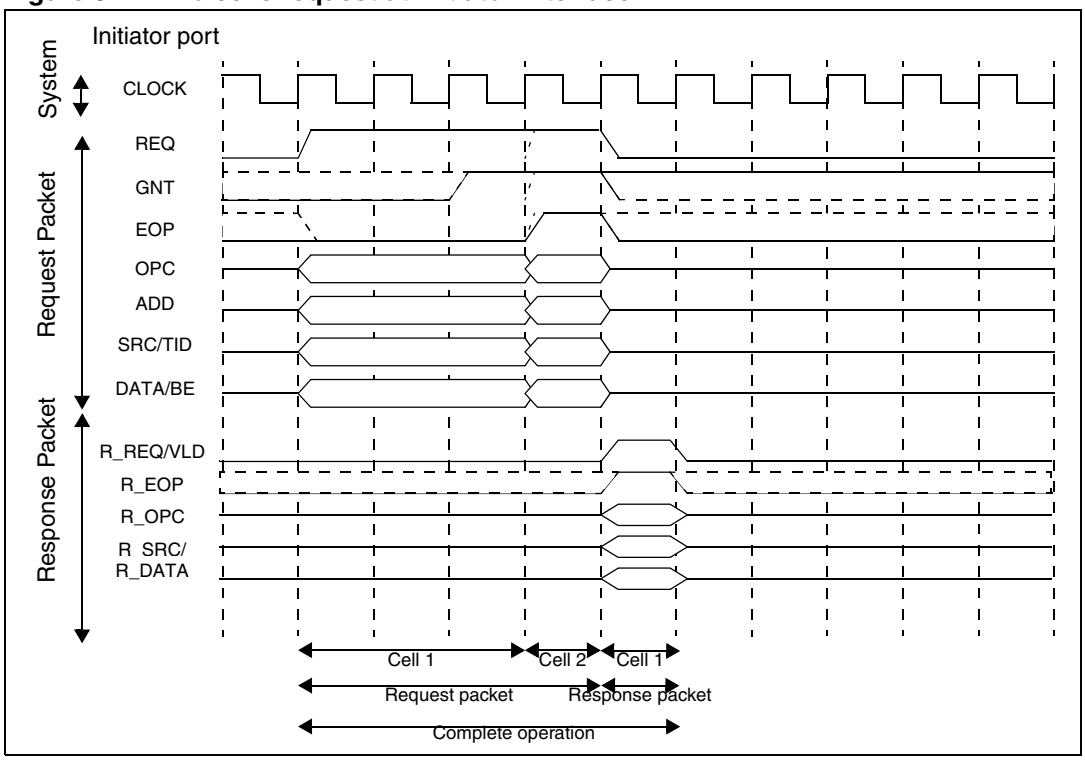
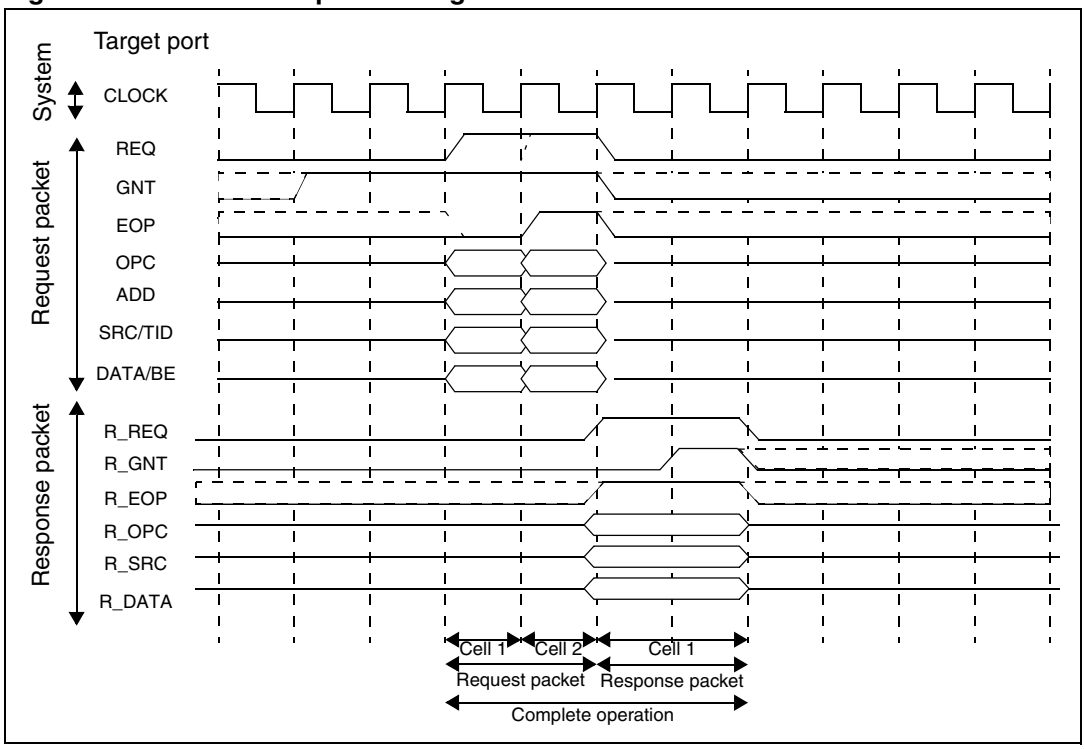


Figure 32. Two cells request at target interface



Four cells request

Figure 33. Four cells request at initiator interface

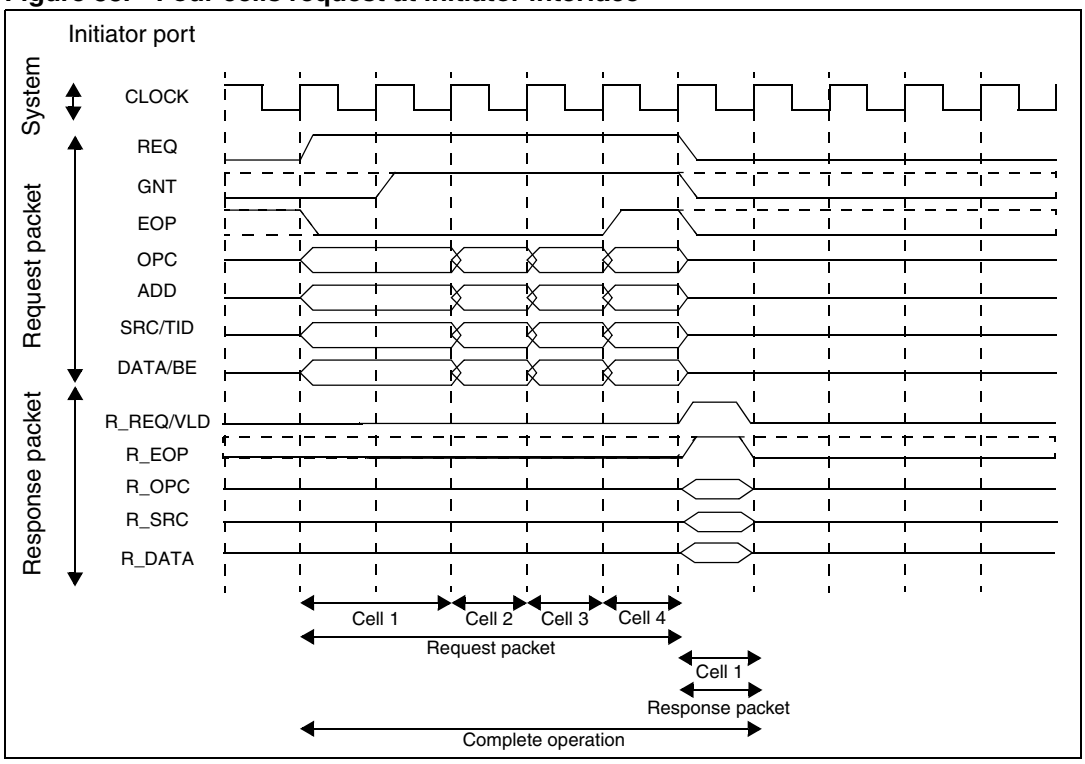
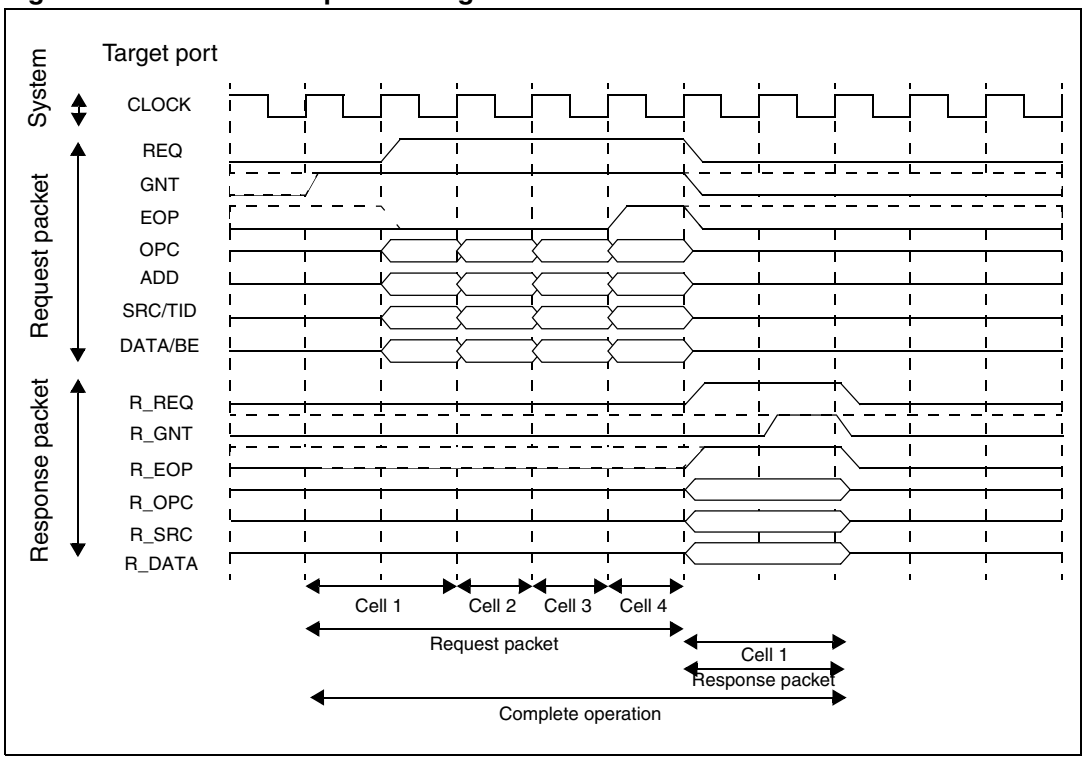


Figure 34. Four cells request at target interface



Two cells RMW

Figure 35. Two cells RMW at initiator interface

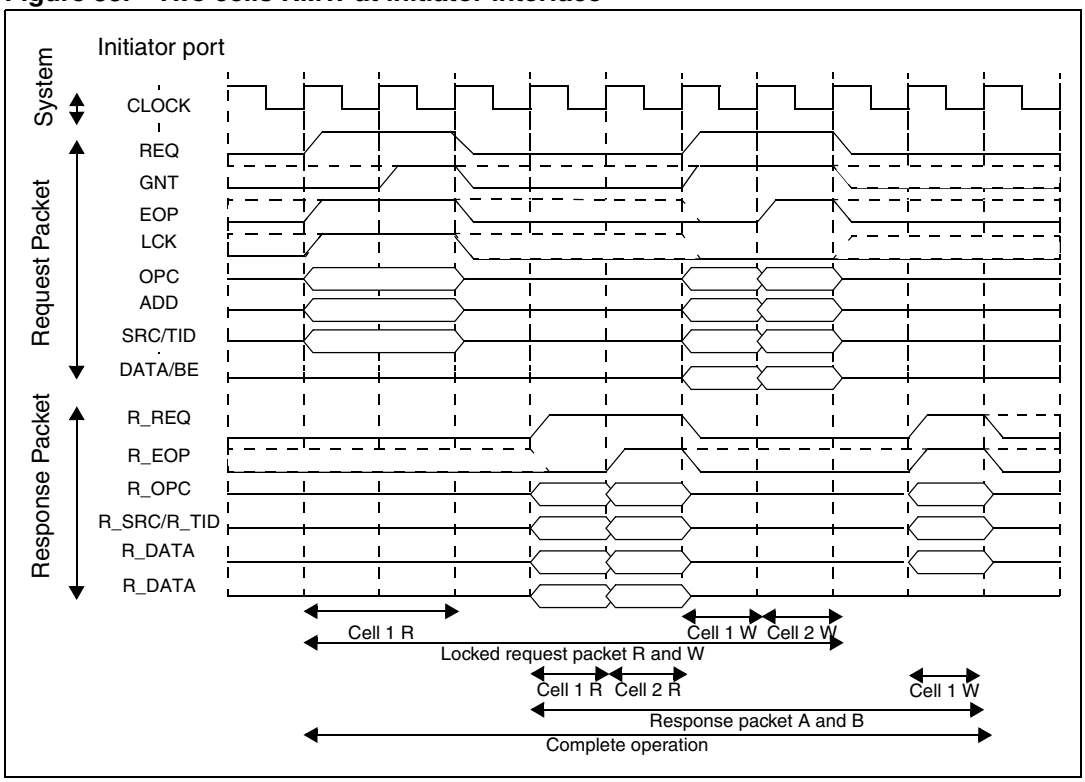
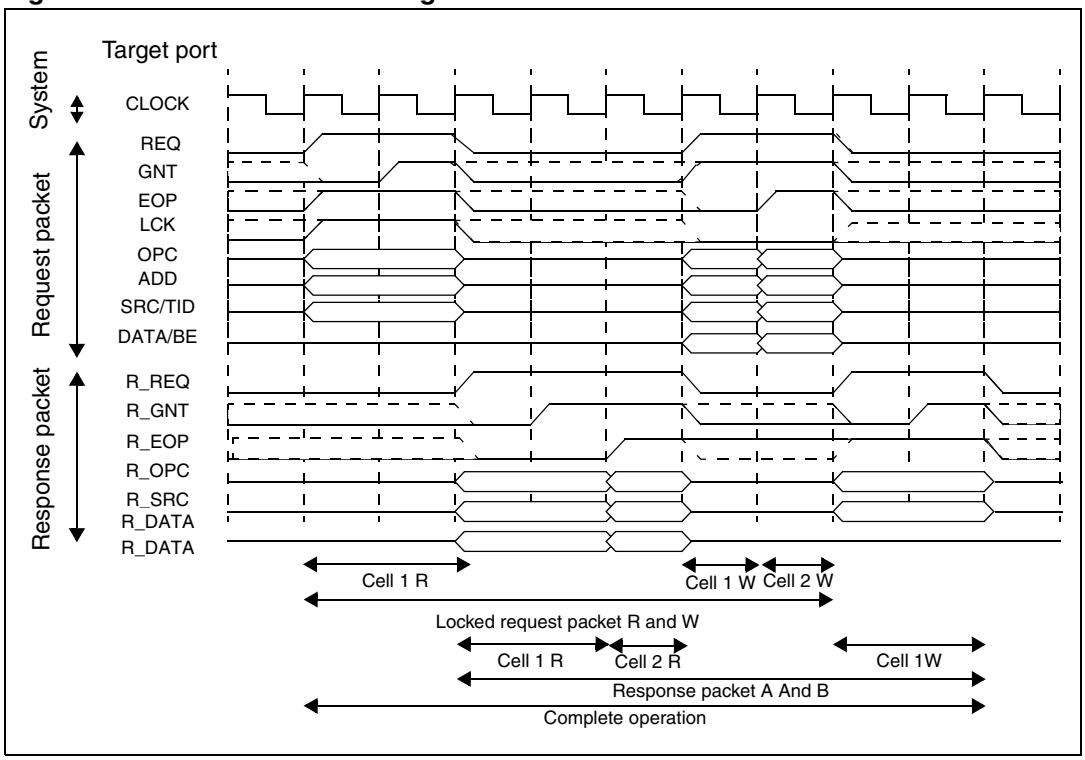


Figure 36. Two cells RMW at target interface



3.4 Error management

3.4.1 Error causes

An error can be generated within an STBus interconnect as a consequence of the following events:

- generation of an illegal address
- generation of an unsupported opcode
- access to switched-off (power down) targets
- security policy violation
- unpredictable events

3.4.2 Error management

The management of an error condition depends on the protocol type of the interconnect domain in which the error occurs, according to the rules in *Table 16*.

Table 16. Comparison of error management between protocol types

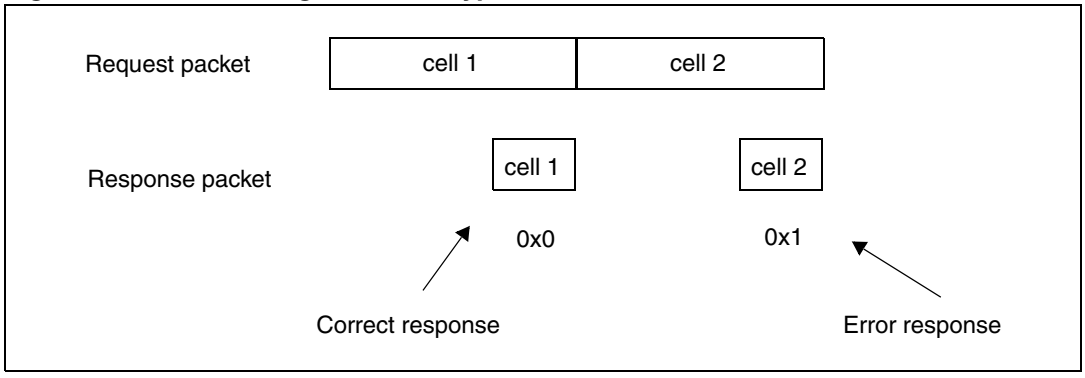
Protocol	Error management condition
Type 1	As type 1 protocol does not support pipelines, each request cell has a response cell associated with it. This response cell is marked by the R_OPC signal which informs of the operation's status. This means that, each request always has immediate visibility of the operation status and all that is required is to analyze the R_OPC field to understand what to do next.
Type 2	Due to the symmetry property of the type 2 protocol, the number of response cells building a response packet always matches the number of cells building the correspondent request packet.
Type 3	Due to the asymmetry of type 3 protocol, it is possible to rely on this flexibility to reduce the number of response cells building a response packet affected by an error. Specifically in type 3, a response packet relative to a load operation affected by an error is composed of the smallest possible number of response cells.

3.4.3 Error management in type 1

From the definitions shown in *Table 16*, it is possible to summarize the error behavior of a type 1 target interface.

- As soon as a request cell has a response cell associated with it (marked by R_OPC = 0x1) an error has occurred, and the type 1 initiator has immediate visibility of that error. It can take this into account for its subsequent processing, or can ask for retransmission of that cell or packet as soon as possible.
- For multi-cell packets (of for example, n cells), if an error occurs in correspondence to the mth cell, all remaining n-m+1 cells are also marked by R_OPC equal to 0x1.

Figure 37. Error management at a type 1 interface



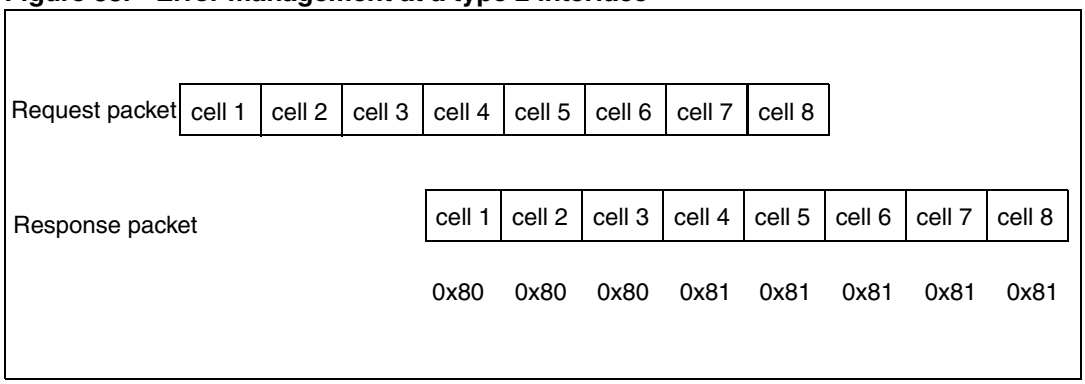
3.4.4 Error management in type 2

From the definitions in *Table 16*, it is possible to summarize the error behavior of a type 2 target interface.

The following behavior must be obeyed when a response packet is composed of n response cells. This is independent of the operation type.

If the error occurs in correspondence with the mth response cell (with m < n), n response cells are sent back with the first m-1 response cell marked by R_OPC equal to 0x80. The remaining n-m+1 response cells are marked by R_OPC equal to 0x81, and the R_EOP signal asserted in correspondence with the last (nth) response cell.

Figure 38. Error management at a type 2 interface

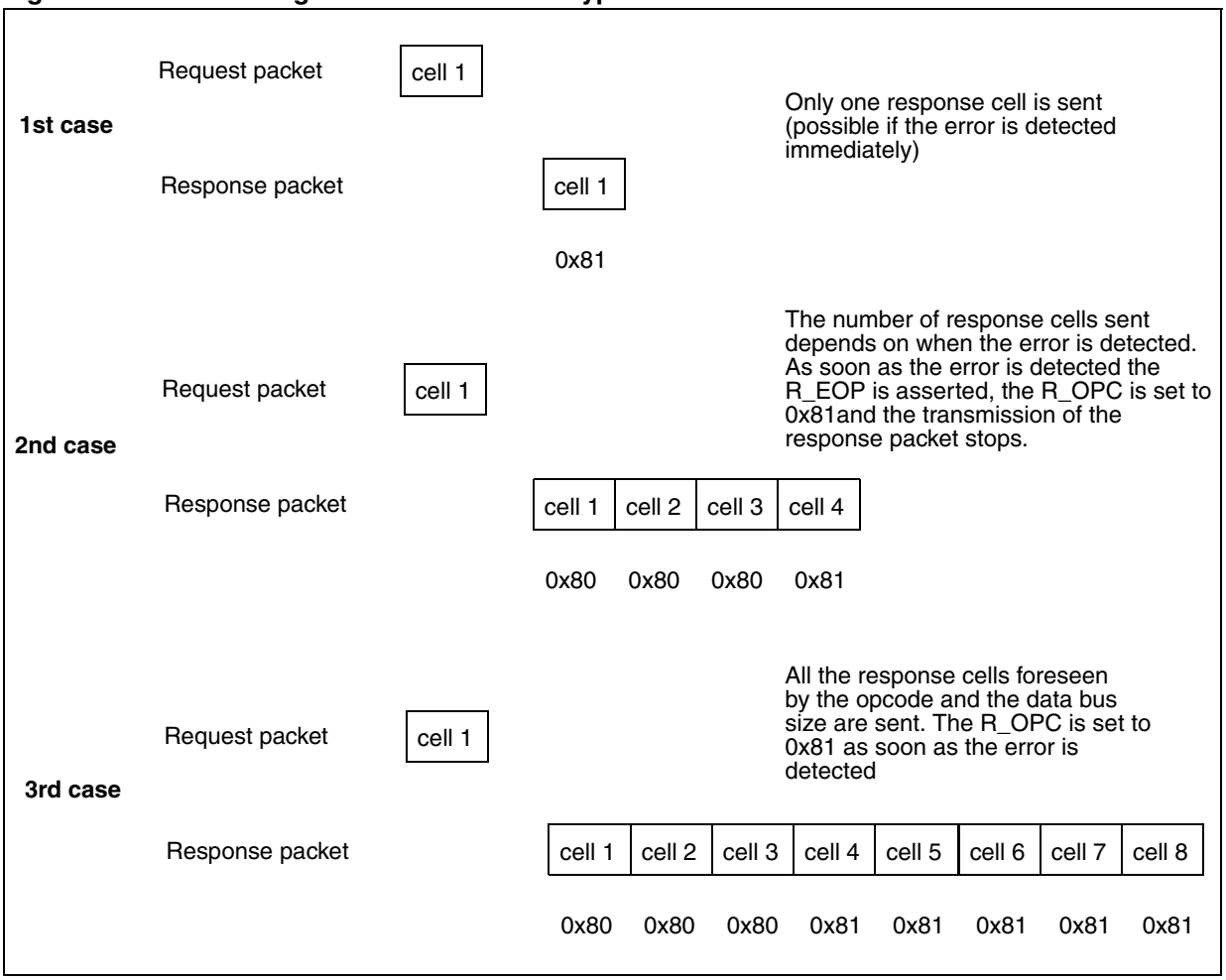


3.4.5 Error management in type 3

From the definitions in *Table 16*, it is possible to summarize the error behavior of a type 3 target interface.

- For a store operation, the response packet is composed of only one response cell, marked with the R_ROP set to 1 and the R_OPC set to 0x81.
- For a load operation, whose response packet should normally be composed of n response cells, there are three different behaviors possible (refer to *Figure 39*).
  - Only one response cell is sent (typically if it is possible to detect the error immediately).
  - If the error occurs in correspondence with the mth response cell (where m < n), only m response cells are sent back. The first m-1 response cells are sent back with R_OPC equal to 0x80 and the mth response cell sent back with R_EOP asserted and the R_OPC set to 0x81.
  - If the error occurs in correspondence with the mth response cell (where m < n), n response cells are sent back. The first m-1 response cells marked by R_OPC equal to 0x80, the remaining n-m+1 response cells marked by R_OPC equal to 0x81. The R_EOP is asserted in correspondence with the last (nth) response cell.

Figure 39. Error management behavior with type 3 interface



3.4.6 Behavior of the STBus building blocks with respect to errors

The behavior of all the STBus building blocks should be consistent with the description given in *Section 3.4.3* to *Section 3.4.5*, and with the following rule.

The STBus interconnect must not perform any processing on the response packets. In case of response packet errors, the number of response cells received from the target interface is propagated back to the initiator together with their size and type.

In case of errors generated by a type 3 node (due to wrong address, security violation or access to targets in power down mode) only one cell response packet is generated.

Appendix A Glossary

Arbitration	The process used to decide which initiator can take possession of the system buses depending on priority and the implemented (or selected) arbitration scheme.
Buffer	A buffer is a retiming block used to connect together two different STBus nodes. A buffer acts as a target for the first node and as an initiator for the second node. It can be also used as generic retiming stage when retiming is needed.
Cell	A cell is the most basic data information which can be transferred along the bus within a single clock cycle; its size equals the data bus size.
Chunk	A chunk is a set of packets linked together by the LOCK (LCK) signal. Its last packet is marked by the LCK low. A chunk is not interruptible.
Frequency converter	A frequency converter is a block acting as an adapter between two blocks working at different clock frequencies to each other, when the two clocks are completely asynchronous or when a phase relationship between them exists (semi-synchronous mode).
Initiator	An initiator is a device accessing the system resources through the STBus node; an initiator port sends request packets and receives response packets.
Message	A message is a collection of chunks linked together by the TID[4] signal (see *Section 2.2: Transaction signals on page 10*). The last chunk in a message is marked by TID[4] low. A message is interruptible if a higher priority initiator is performing a request. This guarantees the initiator transferring the message has the required bandwidth and also guarantees a low latency to the initiator.
Node	The STBus node is the simplest interconnect system following the STBus protocol, to which ports of the same data size are connected. An STBus node is composed of two main blocks: the control and the datapath. The former is the block performing the arbitration, the latter is the routing network through which data flows.
Packet	A packet is a set of cells and its last cell is characterized by the end of packet signal (EOP) high. A packet is not interruptible.
Protocol	A protocol is a set of rules that initiators and targets follow during a transaction. These rules are expressed in terms of signal management (asserting and de-asserting) during a transaction.
Register decoder	The register decoder is a block allowing one or more type 1 initiators to interface with a number of type 1 targets, usually responsible for register accesses. If there are more than one type 1 initiators, the register decoder can be seen as a type 1 node, since an arbitration mechanism is implemented within it.

Size converter	A size converter is a block acting as an adapter between two different systems (such as STBus nodes) each having a different data bus size.
STBus	STBus is a set of protocols, interfaces, primitives and architectures specifying an interconnect subsystem, versatile in terms of performance, architecture and implementation.
Target	A target is a resource for the system which is accessed by initiators through the STBus node: a target port receives request packets and sends response packets.
Transaction	A transaction is an exchange of information between an initiator and a target. It can consist of a request packet and the relative response packet, a request chunk and the relative response chunk, or a request message and the relative response message.
Type converter	A type converter is a block acting as adapter between two blocks following different STBus protocol types (such as an initiator and a node or a node and a target).

Revision history

Table 17. Document revision history

Date	Revision	Changes
19-Nov-2007	1	Initial release.
04-Oct-2012	2	Removed erroneous watermark.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY TWO AUTHORIZED ST REPRESENTATIVES, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com