

# MBus Specification

Pat Pannuto <ppannuto@umich.edu>

Revision 0.2\* — May 4, 2013

## Overview

MBus is an ultra low power system bus. The original design was motivated by the Michigan Micro Mote (M3) project. The goal of MBus, however, is to be a general purpose bus for hyper-constrained systems. MBus requires four pins per node, uses purely digital logic, supports arbitrary length transfers, and features a low-latency priority channel and robust acknowledgments. MBus member nodes do not require a local clock and are capable of completely stateless operation. MBus requires one more powerful node to act as a bus *master* node, whose primary duties are providing the MBus clock and mediating arbitration.

## Contents

<b>1</b>	<b>Requirements &amp; Design Considerations</b>	<b>3</b>
1.1	Reset Design	5
1.1.1	The Reset Detector	6
1.1.2	Hung Nodes	6
1.2	Power Gated Nodes	6
1.3	Design Caveats	7
1.3.1	Broadcast Message “Acknowledgment”	7
1.3.2	Interruptions: Arbitrary Length Messages and Forward Progress	7
<b>2</b>	<b>Node Design</b>	<b>9</b>
2.1	Physical Design	9
2.1.1	Member Nodes	9
2.1.2	Master Node	9
2.1.3	Bus Connections	9
2.1.4	Injection	10
2.2	Logical Design	10
2.2.1	Forwarding	10
2.2.2	Transmitting	11
2.2.3	Receiving	12
2.2.4	Exception States	12
<b>3</b>	<b>Bus Design</b>	<b>13</b>
3.1	Bus Idle	13
3.2	Arbitration	13
3.3	Message Transmission	14
3.4	Interrupt	14
3.4.1	Nesting Interrupts	16
3.5	Control Bits	16
3.6	Return to Idle	17

<b>4</b>	<b>Power Design</b>	<b>18</b>
4.1	A Brief Background on Power-Gating . . . . .	18
4.2	Waking the <b>Bus Controller</b> . . . . .	18
4.2.1	Handling Interrupts During Wakeup . . . . .	18
4.3	Waking the Node . . . . .	19
4.3.1	Handling Interrupts During Wakeup . . . . .	19
4.4	Waking via Local Interrupt . . . . .	19
4.5	Sleeping the <b>Bus Controller</b> , Node . . . . .	20
4.6	The Master Node, In Brief . . . . .	20
<b>5</b>	<b>Addressing Design</b>	<b>21</b>
5.1	Address Types . . . . .	21
5.2	Full Prefix Assignment . . . . .	21
5.3	Short Prefix Assignment . . . . .	21
5.3.1	Static Short Prefix Assignments . . . . .	22
<b>6</b>	<b>Specifications</b>	<b>23</b>
6.1	Granularity . . . . .	23
6.2	Endianness: Byte-Level . . . . .	23
6.3	Endianness: Bit-Level . . . . .	23
6.4	Minimum Message Length . . . . .	23
6.5	Minimum Maximum Message Length . . . . .	23
6.6	Retransmission . . . . .	23
6.7	Minimum Buffer Size . . . . .	23
6.8	Buffer Overflow / Flow Control . . . . .	23
6.9	Clock Speed . . . . .	24
6.10	Addressing . . . . .	24
6.11	Unassigned Short Prefix: <b>0b1111</b> . . . . .	24
6.12	Interrupt Rules . . . . .	24
6.13	Failed Arbitration (Spurious Wakeup) . . . . .	24
<b>7</b>	<b>Control and Configuration Messages</b>	<b>25</b>
7.1	Broadcast (Address <b>0x0X</b> , <b>0xf000000X</b> ) . . . . .	25
7.1.1	Broadcast Messages and Power-Gating . . . . .	25
7.1.2	Broadcast Channels and Messages . . . . .	26
7.2	Extension (Address <b>0xffffffff</b> ) . . . . .	29
<b>8</b>	<b>ToDo</b>	<b>30</b>
8.1	Future Extensions . . . . .	30
8.1.1	Resuming Interrupted Transfers . . . . .	30
<b>9</b>	<b>Document Revision History</b>	<b>31</b>

# 1 Requirements & Design Considerations

This section attempts to explain the design tradeoffs made in the MBus design.

**Define bus PHY, MAC and application logic interface** We take a vertical integration approach, controlling more of the complete stack in an effort to obtain every available ounce of energy efficiency within our further design constraints.

**Must be fully-synthesizable** In an effort to make MBus ubiquitous, it is important that other designers can easily add MBus to their systems/chips. In practice, this means a fully-synthesizable design such that simple, pure Verilog can be given to any individual wishing to implement MBus. This provides added advantages in pre-silicon testing and validation.

**Optimize for synthesis simplicity over speed** The synthesizability is paramount. As example, standard cell libraries contain both positive and negative edge triggered flip-flops, but a dual-edge flip-flop is not available in many processes. It is important then that no element of MBus design require an implementation that double-clocks flops.

**Must be low-power, low-leakage** The MBus was designed for the Michigan Micro Mote (M3) project. The system is designed to run off a 0.5  $\mu$ Ah battery. To serve as a feasible communication bus, the energy consumption must be on the order of  $pW$ .

This extreme power constraint eliminates traditional analog open-collector style designs as options. Sufficiently low-leakage circuits would have untenable drive strength requirements and/or rise time constraints. The previously stated portability requirements eliminate custom crafted analog elements as an option, restricting MBus design to a purely digital approach.

**Support Multi-master operation / bus arbitration / avoid race conditions** As every layer is capable of entering an extreme low-power state, any layer must be capable of waking the system. The waking layer must also be capable of communicating (i) who woke the system and (ii) why. While such a system could query (ii) given (i), determining who woke the system requires a complex dedicated controller, or for the waking layer to simply announce it. A multi-master bus greatly simplifies the software design and allows for things such as DMA from an imager chip to a radio chip without necessarily requiring CPU intervention.

Once multi-master is agreed upon, some form of arbitration scheme must be devised, as well as consideration for races as multiple layers attempt to acquire the bus.

**Minimize pins**

**Minimize gates**

**Limit the number of I/O pads** The physical area of the M3 system is extremely constrained. There is only room for four I/O *pads* on each layer of the system (keep in mind that forming a net/bus still requires two *pads* when wirebonding).

**Minimize timing uncertainty/latency/jitter**

**Allow for wakeup that takes time** As the system supports extremely low-power sleep, the bus must tolerate / account for a lag in waking layers of the system. In practice this is mostly a focus on the bus layer controller for each layer at first, followed by layer-specific concerns of how much local buffer should be necessary and when/how to wake the proper layer.

**Active vs sleep mode; want backward compatibility** The bus needs an idle state that consumes very little power. Ideally, each layer not participating in the current transaction also burns very little power.

There should also be some consideration for future-proof design, that is the introduction of a newer protocol that can run on the bus that older controllers can silently ignore.

**Ensure that bus reset is possible** Stuff happens. It is important that there exists some kind of escape hatch to rescue / reset the bus. For deployed systems with multi-year lifetimes, the ability to reset to a known state from nearly any possible erroneous state becomes critical. Details of reset design are discussed in [1.1 Reset Design](#).

**Support variable-length packet sizes** Short packet length limits (e.g. order 4 bytes) lead to unacceptably high protocol overheads for large (order 1 kB+) messages. An upper bound on packet-length leads to possibly complex software-level fragmentation schemes (or worse hardware-level if the bus packet length < message buffer).

A leading length field with a reasonably large upper bound would then be acceptable, although not ideal. The goal is to allow for true arbitrary length packets if possible.

**Consider automatic address space management protocol?** There is an extensibility / simplicity trade-off. I2C only allowed for 128 addresses, which for an individual bus instance was seen as more than enough (still pretty true with the loading requirements, but repeaters / extenders make that less true). The real issue though is address space collisions between generic components. To build a plug 'n play bus of any arbitrary pieces requires the whole corpus of parts to have non-conflicting addresses. This motivates extensibility.

Mbus strikes a balance in its address design, using short, 8 bit addresses in the common case but defining a full, 32 bit address to resolve address space collisions. Addressing details are discussed in [6.10 Addressing](#).

**Allow interposing of the bus to read out / insert data from external source** Sniffing pretty much any bus design is easy, but adding devices post-hoc is a harder requirement. There is great motivation for transient connection of devices (programmer, debugger, etc), and also some motivation for the ability to permanently add new devices to an existing bus.

As Mbus employs a ring style connection, injecting a new device in an existing bus may not be trivial. Interposition of external devices is not seen as a requirement for Mbus to define, however, simply as something to caution system implementers as desirable. For one example of bus injection with minimal pad/pin overhead, consult the M3 Implementation document.

**All wires must be uni-directional** Bi-Directional analog hardware is complex and not synthesizable.

**Granularity** Transmissions are required to be modulo 8 bits in length. The motivation for this is largely a function of the arbitrary nature of the last two bits received. It truly arbitrary lengths were allowed, an interruption occurring after 9 bits were transmitted could be any of 0, 1, or 2 bytes long (0 bytes: out-of-band interruption, 1 byte: End of Message where receiver precedes transmitter and thus discards two bits, 2 bytes: End of Message where receiver follows transmitter and thus all 9 bits are valid).

This hypothetical receiver's design is further complicated if the receiving node is (locally) clockless. The bus interface cannot reliably hand off any bytes until it receives the *11th* bit. It cannot reliably learn that it has received two bytes until the Begin Control edge. There are only three bus clocks between that edge and Idle, and only one edge until the receiver would be obligated to ACK or NAK, a very challenging design constraint to occasionally omit up to 7 bits from a message.

## Endianness

**Byte-Level** No specific design consideration here per se. Given that Mbus supports messages of an arbitrary length, it simply felt more logical to send bytes from 0...N instead of N...0.

**Bit-Level** Again an arbitrary decision. The first test implementation elected to use MSB and the decision was made.

**Retransmission** Hardware retransmission carries risks of accidentally locking up the bus, endlessly retransmitting. While ideas such as a maximum retry count (SMBus) mitigate this, the added complexity and risk of endless hardware retransmission tip the scales in favor of pushing the retransmission decisions to software.

To provide the software layer with more information, we choose to require the indication of the number of bytes actually transmitted. While transient faults could certainly occur, it is reasonable that the software can infer different *probable* causes from the amount of the message that was transmitted.

**Flow Control** Any form of flow control requires communication *mid-transmission* between the transmitting and receiving node. This could be accomplished by periodically (e.g. every word) switching roles, but this introduces a large amount of complexity. Another solution is to enable some form of clock stretching, but these designs are either analog in nature (e.g. I2C) or prohibitively complex to implement.

As MBus supports arbitrary interruptions, a simpler design is possible. If a receiving node's RX buffer is overrun, it interrupts the bus and indicates a buffer overrun. Not interrupting the bus acts as an *implicit* flow control. In practice, transmitting nodes should consider sending some form of *ping* message to a destination node to validate its presence before sending a large transmission to minimize waste if the destination node is not actually present. MBus is not designed for a changing topology, as such discovery of this nature need only be performed once and the amortized cost is considered negligible.

**Acknowledgment granularity** As MBus supports messages of arbitrary length, a natural question arises for the granularity of acknowledgments. At one extreme, designs such as I2C elect to acknowledge every byte. MBus takes the opposite approach, providing only a single acknowledgment at the end of a transmission in an all-or-nothing fashion. The rationale for this decision again returns to simplicity and efficiency in design. While a receiver-driven acknowledgment cycle could easily be added every *nth* bit, the relative merits of this are not great. Assuming a receiver exists on the bus, by not electing to interrupt transmission, a receiver implicitly ACKs every bit sent. A receiver that has occasion to NAK a transmission may do so at any time during transmission. The obvious weakness of the implicit ACK is an absent receiver, however MBus expects a static topology and thus this is not considered an important consideration.

**Streaming** Some bus designs incorporate a “streaming” mode, which allows for a series of physical bus transactions to be considered one contiguous message at the receiving layer. As MBus allows for arbitrary length messages, such a feature becomes largely unnecessary, and MBus defines no streaming primitive.

**Avoid ratioed-logic, avoid timing constraints**

**Idle State Should Be High** For designs that aggressively power-gate components, it is easier to build designs that pull low in minimal power modes than pull high. As a consequence, the Idle state in MBus elects to leave all lines high.

## 1.1 Reset Design

The reset mechanism is the MBus escape hatch. Its reliability is critical such that no matter what state MBus nodes are in, they can all be reliably forced back to a known state. Special care must be taken that no assumptions are made about the state of any node in the system. As example, in normal operation only one node should ever not be forwarding. The reset mechanism, however, must correct for more broken cases where any *N* nodes are not forwarding and return all nodes to a common and known state.

### 1.1.1 The Reset Detector

Mbus's reset detector is advantageous in that it is completely isolated from the Mbus state machine. To reset Mbus, the Interrupt sequence is entered. This entry saturates, allowing nodes to be 'endlessly-interrupted' until all of the nodes have been interrupted. A misaligned state machine, or a timing glitch, or other failures related to normal operation cannot affect the interrupt entry detection, as the detector is an isolated, dedicated circuit. The exact requirements for the Interrupt detector are discussed in [3.4 Interrupt](#).

### 1.1.2 Hung Nodes

We define "hung" as an erroneous state from which a node would never depart without external intervention. We are not concerned with how a node got into an erroneous state—a pathological series of SEUs, whatever—the focus is on how a node excises itself.

**Busy During Bus Idle** In this state, the Mbus is Idle, but a node believes there is still a transmission taking place and will wait forever for a new clock edge, constantly reporting the Mbus as busy.

There is a small window where it is possible to enter this state as a node must error during one of the bits in the control sequence. A node will be stuck in this state until another transmission occurs. The behavior of an erroneous node is not well-defined.

*Aside:* A node with its own local sense of time could include a mechanism that bounds the period of time without a pulse on the clock or data lines, but such a mechanism is outside the scope of this document. The Mbus neither prohibits nor requires clock stretching. A minimum clock period is specified as a function of Mbus topology, but a maximum is not. If building a design with such an escape mechanism then, designers must be conscious of the implementation details of its master node.

**Endless TX** If a transmitter enters an error state and never interrupts the bus, other nodes will not be able to distinguish an endless transmission from a transmission with a long string of 0's (or 1's). Mbus relies on the controller to detect the error. While Mbus does allow for arbitrary message lengths, a master node must define a maximum message length for a particular Mbus instantiation<sup>1</sup>. The master node counts the number of bits received and can forcefully reset the bus if necessary. Address bits count towards this limit. Interrupt entry edges do not count towards this total as the rising clock edges will not be seen by the master node's CLK\_IN. The intention of the limit is not to be used ever during correct Mbus function, and thus should be set well above the maximum expected message size. The absolute minimum limit for a conforming Mbus is 1 kb ( $[1024 - \text{addr\_len}]$  data bits).

## 1.2 Power Gated Nodes

As Mbus is designed for extremely low-power systems, it is reasonable to address the needs for the coordinated wake-up of power gated nodes. A power gated device requires a series of well-organized steps to awake in a known state. This waking is much like a standard chip power-on-reset sequence, only Mbus design seeks to avoid imposing custom delay elements on the construction of a node's bus controller.

The following sequences are designed for nodes that are possibly receiving the transmission. Waking the transmitting node is left to alternative means by the implementation, with the tacit assumption that if a node has something it wishes to transmit, it is probably already awake<sup>2</sup>.

The generalized wake-up sequence for power-gated devices requires up to four signals:

- Power on (remove power-gating)
- Warm up local oscillators
- De-assert Reset

<sup>1</sup> It is suggested that this value be programmable, for maximum flexibility.

<sup>2</sup> Though, a design that wished to exploit these pulses could take advantage of the spurious wakeup rules described in [6.13](#). Allow the first transmission to fail and use the available pulses to wake itself. This design would still require some care, as the node would have to begin forwarding its data line again before the arbitration pulse. Using the falling edge of the clock line as the transition from pulling data low to forwarding would be sufficient to achieve this goal.

- Remove isolation

Power gated MBus member nodes are able to use the bus's clock line to provide all four of these edges. The first four positive clock edges of a MBus transmission are: arbitration, priority drive, priority latch, and begin transmission, none of which require receiver intervention. The fifth clock edge latches the first bit of the transmitted address. The implication is that a node using MBus clock edges to wake from power-gating must reset into a state that is prepared to receive the first address bit.

Sleeping a node is a simpler process, requiring only two signals:

- Isolate device
- Power off (power-gate)

By the time a node is sending (or forwarding) the second control bit, it has decided whether it will be putting itself to sleep or not. If a node intends to sleep, it can use the clock pulse that latches the second control bit and the pulse that formally transitions the bus to idle as the two required edges.

### 1.3 Design Caveats

Here we attempt to draw attention to some of the systems-level issues that may be presented to persons using MBus.

#### 1.3.1 Broadcast Message “Acknowledgment”

For point-to-point messages, MBus acknowledgments provide a strong guarantee to higher layers:

*The message was received in full or the message was not received at all.*

For broadcast messages, the MBus semantics are weaker:

*The message was received in full by at least one node or the message was not received at all by any node.*

Systems cannot rely on a “successfully” broadcast transmission as *guaranteeing* that every node has received the message. If an absolute guarantee is required, systems must build one atop point-to-point messages.

#### 1.3.2 Interruptions: Arbitrary Length Messages and Forward Progress

MBus design provides a powerful tool: the ability for any node to interrupt any message at any point. Such a mechanism invites risk of live-lock on the bus, however.

As a first example, consider two nodes that simultaneously wish to send an interrupt-worthy high-priority message. One node will win arbitration and begin transmitting, only to be immediately interrupted by the other. The two nodes will continuously interrupt one another in an effort to send their very high-priority message, and neither will ever succeed.

Another form of live-lock stems from periodic low-latency messages. Consider a system with a timer chip that needs to generate periodic 100 ms pulses that another chip will need to respond to in a bounded latency. The responsiveness requirement will place an interrupt-class priority message on the bus every 100 ms. On a 400 kHz instantiation, if another node (say a camera node) attempts a bulk transfer (an image) it will only be able to send approximately 40 kilobits before being interrupted. A picture over 5 kB will never send.

As the only physical bus available for a complex system, MBus is required to support both timing-critical messages and large data transfers. To provide sufficient flexibility to efficiently handle both types of messages while maintaining very simple node architectures (e.g. no suspending transfers), MBus pushes additional considerations to the systems design.

MBus provides both some requirements and some guidelines to help system designers:

**Requirements** These are elements of the MBus specifications designed to help systems with bus contention issues.

- The first 32 bits of a message may not be interrupted

- There is no means in MBus arbitration to ensure all nodes can distinguish priority and regular messages<sup>3</sup>. As such, MBus guarantees that at a minimum 32 bits of data may be transmitted without interruption (6.12 Interrupt Rules).

**Guidelines** These guidelines operate under the assumption that a priority message is of sufficiently high priority that it would warrant interrupting an active transmission. Without preemption, the livelock scenarios are averted, however the issues of starvation remain.

- Priority Message Length
  - Priority messages are permitted to be any length, *however*, only the first 32 bits—even for priority messages—are protected from interruption. A design that sends two interrupt-worthy high-priority messages longer than 32 bits *will* livelock.
  - MBus strongly *recommends* that all priority messages be restricted to 32 bits. With great care and caution designers may violate this constraint.
- Priority Message Frequency
  - As priority messages also have a topology-dependent component, MBus advises that after sending a priority message a node back off from sending *any* message (regular or high priority) for at least 80<sup>4</sup> bit-times per MBus node.
  - This is not a formal MBus requirement as it is a recognized design constraint that not all member nodes are required to have any sense of time. Furthermore, it is believable that such time-less nodes may be sensors that require a low-latency response. It is not the intention of a bus design to define how such an issue is dealt with. It is, however, then the obligation of the designer of such a sensor to carefully consider how to be a good MBus citizen and how to avoid saturating the system’s only communication bus.
- Normal Message Length
  - On a system with interrupts
    - \* The first concern must be successful forward progress in the face of interrupts. Long messages must be packetized (most pathologically down to 32 bit windows, less headers). Future MBus designs consider more amenable solutions to this drawback, see: 8.1.1 Resuming Interrupted Transfers.
    - \* In practice, consider the shortest interval of interrupting messages your system may encounter. Multiply this window by the bit time and halve the resulting value to get an approximate maximum packet size.
  - On a system without interrupts (or with infrequent ones)
    - \* Long messages cause starvation. You must consider the acceptable average latency and use this coupled with the bus speed to define a reasonable maximum message size.
    - \* Frequent short messages can cause starvation. While nodes are not required to have a sense of time, those that do should back off for a window of time between messages. Nodes without a sense of time should be designed with avoiding flooding in mind.

<sup>3</sup> e.g. in Figure 4, Node 2 would have seen the exact same signals if Node 3 had not participated at all.

<sup>4</sup> Arbitration (4) +  $t_{long}$  (2) + address (8 or 32) + data (32) + interrupt (6) + control (2) + idle (1) = 79.



## 2 Node Design

The MBus defines two *physical* types of nodes: member nodes and a master node. An instantiation of MBus must have one and only one master node and must have at least one member node ( $N \geq 1$ ). The maximum number of member nodes is a function of clock speed<sup>5</sup>.

During a transmission, MBus defines three *logical* types of nodes: a transmitting node, a receiving node, and forwarding nodes. During a transmission, there must be exactly one transmitting and one receiving node. Any number ( $N \geq 0$ ) of forwarding nodes are permitted.

### 2.1 Physical Design

From a package perspective, the physical design of a member and master node is the same, each must expose a DIN, DOUT, CLKIN and CLKOUT pad.

In addition, one (or more?) chip(s) on the bus can add two additional pads to act as a “splitter” node, to allow the interjection of new devices on the bus. This part of MBus is not yet well-defined.

#### 2.1.1 Member Nodes

A member node requires 4 signals:

- DIN – Data In
- DOUT – Data Out
- CLKIN – Clock In
- CLKOUT – Clock Out

Most of the time, a member node is in the FORWARDING state. While forwarding, a member node must amplify and forward signals from the DIN pin to the DOUT pin and from the CLKIN pin to the CLKOUT pin. Designs should attempt to minimize latency between these pins. The maximum propagation latency<sup>6</sup> permitted is 10 ns. MBus defines a maximum load capacitance of XXX pF for the DIN pin and of XXX pF for the CLKIN pin to enable inter-operability of generalized components<sup>7</sup>.

#### 2.1.2 Master Node

A master node requires 4 signals:

- DIN – Data In
- DOUT – Data Out
- CLKIN – Clock In
- CLKOUT – Clock Out

While forwarding, the master node is subject to the same propagation latency constraints (10 ns) as member nodes.

#### 2.1.3 Bus Connections

- DIN, DOUT
  - The data pins shall be connected in a round-robin fashion, the DOUT of one chip connected to the DIN of the next.
  - The connection of data lines must form a loop when connected correctly (e.g. Figure 1).
  - There are no requirements for the placement of nodes in the data loop, but the ordering will have an impact on bus arbitration. See Section 3.2 for more details.

<sup>5</sup> The maximum chip-to-chip latency is defined as 10 ns. Assuming a TX node has similar delay to generate the next bit and clock delay to nodes is 0, then the minimum cycle time is  $N \text{ nodes} * 20 \text{ ns}$ . MBus designers are obligated to consider all possible delays and thus define a maximum feasible clock speed, however, for most designs the maximum clock speed theoretically possible will largely exceed the plausible speed for the given power budget.

<sup>6</sup> The amount of time taken to propagate a change in the output of the *previous* link in the data loop to the output of the next link in the data loop. This time includes all of the internal forwarding logic from DIN to DOUT or CLKIN to CLKOUT, as well as any pin/wire capacitance that must be overcome to drive the next input gate.

<sup>7</sup> However, the only strict requirement is the 10 ns propagation delay, thus careful designers may violate this requirement if necessary.

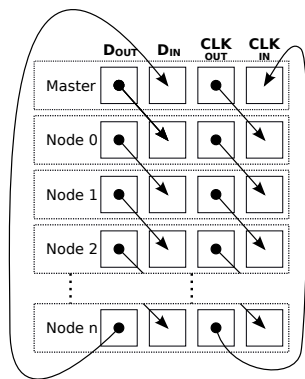


Figure 1: High-level picture of MBus physical design. Member nodes and a master node are connected in a loop, with data and clock lines forming independent rings.

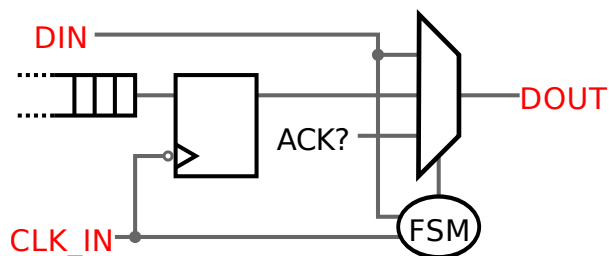


Figure 2: A sketch of the logical model of a node. The Finite State Machine selects between the three modes a node can be in. Top: *forwarding*, Mid: *transmitting*, Bot: *acknowledging*. This model omits some of the subtleties of arbitration, see 2.2.2 for details.

- CLK

- The clock pins shall be connected in a round-robin fashion. The CLKOUT of one chip connected to the CLKIN of the next.
- The connection of clock lines must form a loop when connected correctly (e.g. Figure 1).
- The connection of clock lines must match that of data lines. That is, if a node  $N_a$  connects its DOUT to the DIN of node  $N_b$ , the CLKOUT of  $N_a$  must be connected to the DIN of  $N_b$ .

A MBus must have a minimum of two nodes (one master node and one member node). Single chips that are not connected to a bus should tie CLKIN and DIN high and leave CLKOUT and DOUT floating.

#### 2.1.4 Injection

Injection is the ability for an arbitrary additional node to temporarily (or permanently) interpose itself into a MBus instantiation. Two examples are a system programmer or a debugger.

The exact method of injection is not defined by MBus. It may be as simple as exposing a pair of out/in pins that are normally jumpered, some packages may not have such luxury. Injection is mentioned here as it is an exceptionally useful tool for system bring-up and development. The means for performing injection should be considered as a MBus instantiation is designed.

## 2.2 Logical Design

There are three *logical* types of MBus nodes: transmitting, receiving, and forwarding. MBus can be considered multi-master, any node is capable of transmitting to any other node. The master node adopts these same three personalities, but its behavior differs slightly during arbitration (3.2).

### 2.2.1 Forwarding

Forwarding is the most common state for all MBus nodes. Observe in Figure 2 the very simple, short logic path from DIN to DOUT. Nodes are obligated to forward data in less than 10 ns.

**Forwarding.IDLE** This is the rest/idle state for MBus nodes. In this state member nodes may be completely power-gated and asleep. The only obligation is that DIN is forwarded to DOUT and CLKIN is forwarded to CLKOUT.

*Master Node Exception:* In IDLE, the master node does not forward DIN to DOUT.

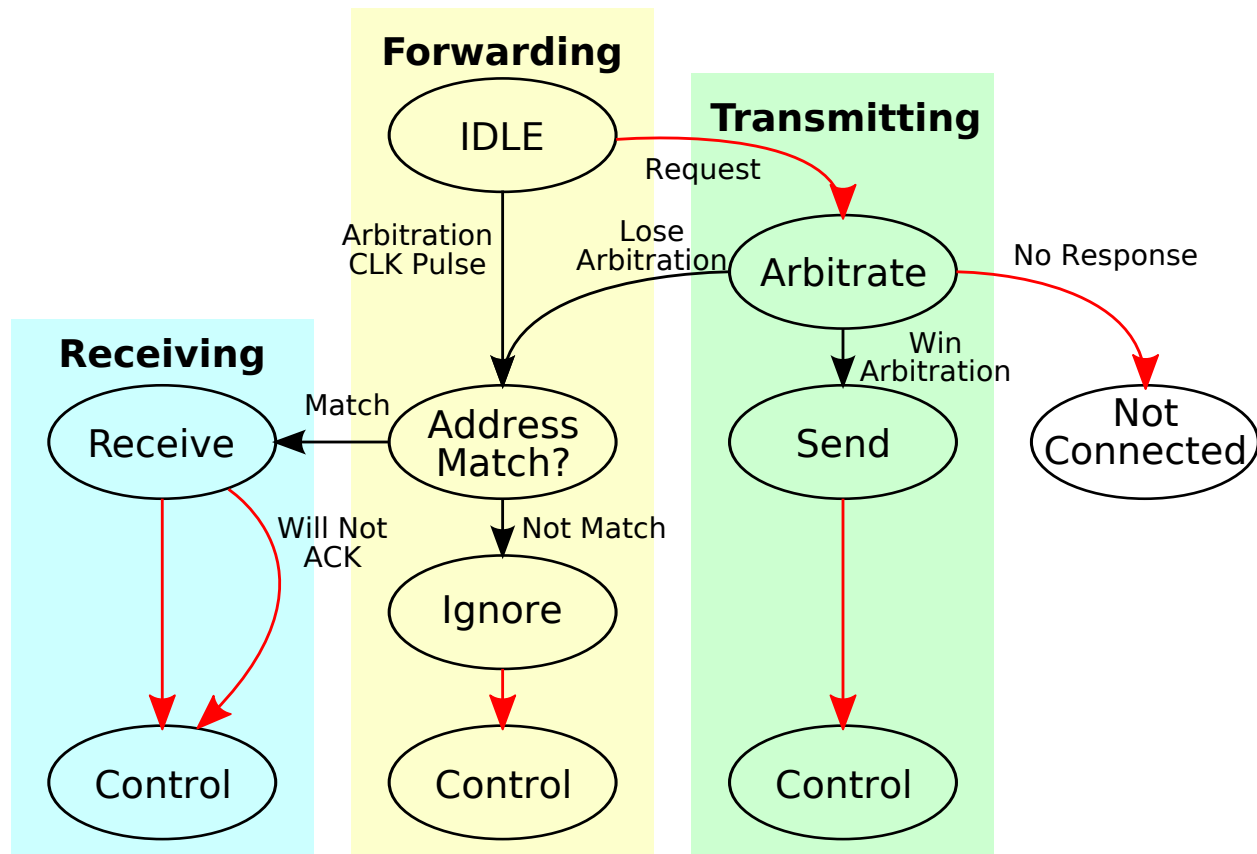


Figure 3: A FSM describing the high-level logical behavior of MBus nodes. Black arrows indicate transitions that occur on bus clock edges. Not shown are implicit arrows from any state to Forwarding.CONTROL. From any CONTROL state, nodes progress to IDLE.

**Forwarding.ADDRESS\_MATCH** At the start of a new transmission, a forwarding node should monitor the DIN line to see if it is the target for this transmission. If a node matches its address, it promotes itself from forwarding to receiving. After the first mis-matched bit a forwarding node transitions to the IGNORE state for the rest of the transaction.

**Forwarding.IGNORE** In IGNORE nodes simply forward data. Nodes remain in this state until Interrupted. Power-conscious designs may safely power gate everything except Interrupt detector circuitry while in IGNORE.

### 2.2.2 Transmitting

**Transmitting.ARBITRATE** A node initiates a transmission by pulling its DOUT line low. A node may only attempt to initiate a transmission while the bus is idle. Care must be taken in detecting the bus idle state when requesting to transmit. In particular, a node requesting to transmit must ensure that the CLK line is still high, it is not sufficient to rely on the local state machine still being in the IDLE state<sup>8</sup>.

The ARBITRATE state is left when the CLK line is pulled high by the master node. If a member node's DIN is high on the rising clock edge, it has won arbitration. A node that loses arbitration should begin listening

<sup>8</sup>To envision the case defended against here, picture a tall stack of nodes, where the bottom node requests the bus. The master node pulls CLK low in response. Shortly before the master node pulls CLK high, a node at the top of the stack (still in the IDLE state) elects to transmit. There is not enough time for his DOUT to propagate to the bottom node, however, thus when CLK goes high, both the bottom and top nodes believe they have won the arbitration. Designers must select a sufficiently long period  $t_{long}$  to ensure all signals are stable. Twice the maximum possible propagation delay (delay for falling CLK to furthest node + furthest node's DOUT back to closest node) of the system should be sufficient.

to see if it is the destination node. If the CLK line never goes high—where never is defined as four times the minimum clock speed of MBus<sup>9</sup>—the node should consider itself as disconnected.

Details of priority arbitration are omitted here for simplicity. See 3.2 Arbitration for details.

*Master Node Exception:* The master node always wins arbitration. Note that when this occurs, the master node's DIN will be low.

*Master Node Exception:* If a master node pulls its DOUT line low and its DIN line never goes low, it should be considered NOT\_CONNECTED.

**Transmitting.SEND** During SEND, a transmitting node pushes bits onto the bus as described in 3.3 Message Transmission.

A transmitting node completes its transmission by Interrupting (3.4 Interrupt) the bus and indicating the transmission is complete.

**Transmitting.CONTROL** As a transmitter, a node is responsible for the first control bit. This bit should be set by a transmitter to indicate that the complete message has been sent. The transmitter must listen to the subsequent control bits to establish if the message was acknowledged and if the targeted layer is sending a response.

### 2.2.3 Receiving

**Receiving.RECEIVE** A receiving node is also obligated to forward data along the bus. In the sketch shown in Figure 2, during the receiving process, a receiving node remains in pass-thru mode until Interrupted.

**Receiving.CONTROL** A receiver must acknowledge the successful receipt of a transmission. If a receiver wishes to NAK a transmission, it simply does nothing during the ACK/NAK control bit.

A receiver may also possibly enter control by electing to Interrupt the bus itself. A receiver will Interrupt a transmission to indicate that its RX buffer has been overrun and the current transmission must be aborted.

### 2.2.4 Exception States

**Exception.NOT\_CONNECTED** A robust implementation should include detection of some kind for an attempt to utilize the bus when the node is not actually connected to a bus (so that it may report failure). After a node pulls its DOUT low there is a maximum possible  $t_{long}$  of **TODO** before a master node must pull CLK low in response. If CLK is not pulled low, the node should consider itself disconnected and report failure to send as appropriate.

---

<sup>9</sup>**TODO: TBD**

### 3 Bus Design

During normal operation, MBus remains in Bus Idle (3.1). A transmission begins with Arbitration (3.2), then Message Transmission (3.3). The transmitter then Interrupts (3.4) the bus and indicates the complete message was sent. During the Control (3.5) period acknowledgment is negotiated. After Control, MBus returns to Idle or may optionally send a response message.

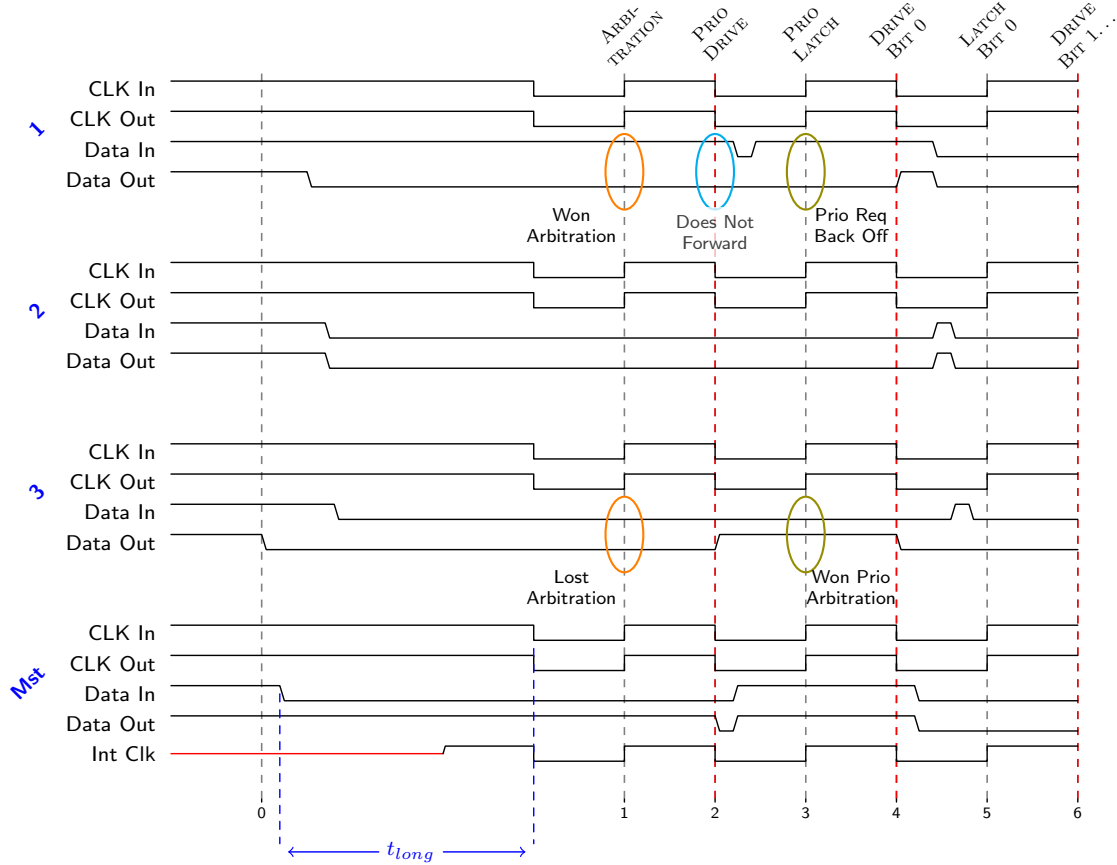


Figure 4: MBus Arbitration. To begin a transaction, nodes pull down on data. Here we show node 1 and node 3 requesting the bus at nearly the same time (node 1 shortly after node 3). Node 1 initially wins arbitration, but node 3 uses the priority arbitration cycle to claim the bus. The propagation delay of the data line between nodes is exaggerated to show the shoot-through nature of MBus.

#### 3.1 Bus Idle

In MBus Bus Idle, all lines (clock and data) are high. All member nodes are in forwarding state and the master node is waiting to begin arbitration.

#### 3.2 Arbitration

To begin arbitration, the bus must be in idle state.

To request to transmit on the bus, a node should pull its DOUT line low. All member nodes remain in forwarding state during arbitration, thus when their DIN line goes low, they are obligated to pull their DOUT line low. The master node, however, does **NOT** pull its DOUT line low in response to its DIN line going low. The master node only pulls its DOUT line low when it wishes to transmit on the bus. When the master node's

DIN line goes low, it will pull the CLK line low and hold it low for some period  $t_{long}$ . During IDLE (and thus arbitration), member nodes must forward the clock signal.

By the end of the period  $t_{long}$ , the effect of the arbitration is achieved. A member node is in one of three possible states:

1. Clock low, DIN high, DOUT high – Lost arbitration (didn’t participate)
2. Clock low, DIN high, DOUT low – Won arbitration
3. Clock low, DIN low, DOUT low – Lost arbitration (either lost, or didn’t participate)

The rising edge of clock that follows  $t_{long}$  commits the results of arbitration and all participating member nodes advance their state machines appropriately.

If the master node wishes to transmit on the bus, all member nodes will be in the third state. Note this arbitration protocol introduces a *topology-dependent priority*. Firstly, the master node has a greater priority than any member node as its DOUT will propagate around the entire data loop. The priority of the member nodes is inversely related to their proximity to the master node in the data loop. That is, the furthest node from the master node, the node whose DIN is connected to the master node’s DOUT, has the highest priority of member nodes. The closest node to the master node, the node whose DOUT is connected to the master node’s DIN, has the lowest priority.

At the end of  $t_{long}$ , the master node drives the clock high. The normal arbitration phase ends on this rising edge. The next two clock edges allow for a high-priority arbitration. As MBus priority is topology-dependent, we provide a priority arbitration cycle to allow a low priority member node to preempt transmissions. This priority mechanism is still topology-dependent, that is, if Node 1 in Figure 4 had also attempted to send a priority message, it would have won that arbitration as well. The priority arbitration cycle is two clock edges (one clock cycle), the falling edge after arbitration is for nodes to drive their priority requests onto the bus and the rising edge is used to latch the results. During priority arbitration, the node that won the original arbitration **must not** forward its DIN to DOUT. At the end of priority arbitration, if the original winner did not request a priority message, the node must sample its DIN line. If the line is still low there was no priority message, if the DIN line has gone high, however, the node has lost priority arbitration and must back off to allow the priority message requester to transmit.

Whichever node ultimately won arbitration transitions from a forwarding node to a transmitting node. Nodes that lost arbitration revert to Forwarding.ADDRESS\_MATCH.

### 3.3 Message Transmission

The falling edge after arbitration is defined as “Begin Transmission”. On this edge, the transmitting node should switch its DOUT mux from forwarding DIN to the transmit FIFO. Data is expected to be valid and stable during every subsequent rising clock edge.

The first sequence of bits pushed onto MBus are the *address* bits. All nodes on a MBus should listen until their address:

- Matches: Node promotes itself from Forwarding.ADDRESS\_MATCH to Receiving.RECEIVE.
- Does Not Match: Node transitions from Forwarding.ADDRESS\_MATCH to Forwarding.IGNORE.

There is no protocol-level delineation between address and data bits. The transmitting node sends address+data as a continuous stream of bits (for details on MBus addressing, see 6.10 Addressing). Once a transmitter has sent the complete transmission it Interrupts the bus.

Legal transmissions on MBus must be byte-aligned (6.1 Granularity). The requirement allows receiving nodes such as node 1 and node 3 in Figure 5 to disambiguate the significance of the last two bits received. A legal transmission will end cleanly on a byte boundary if the node topologically follows the transmitter (e.g. node 3) or will end on a byte+2-bit boundary if the node topologically precedes the transmitter (e.g. node 1).

### 3.4 Interrupt

During normal operation, Interrupt is only permitted after the first 32 bits of data have been transmitted (6.12 Interrupt Rules). The Interrupt mechanism, however, is also the MBus reset mechanism, and as such member node interrupt detectors **must** always be active whenever a node is not Idle.

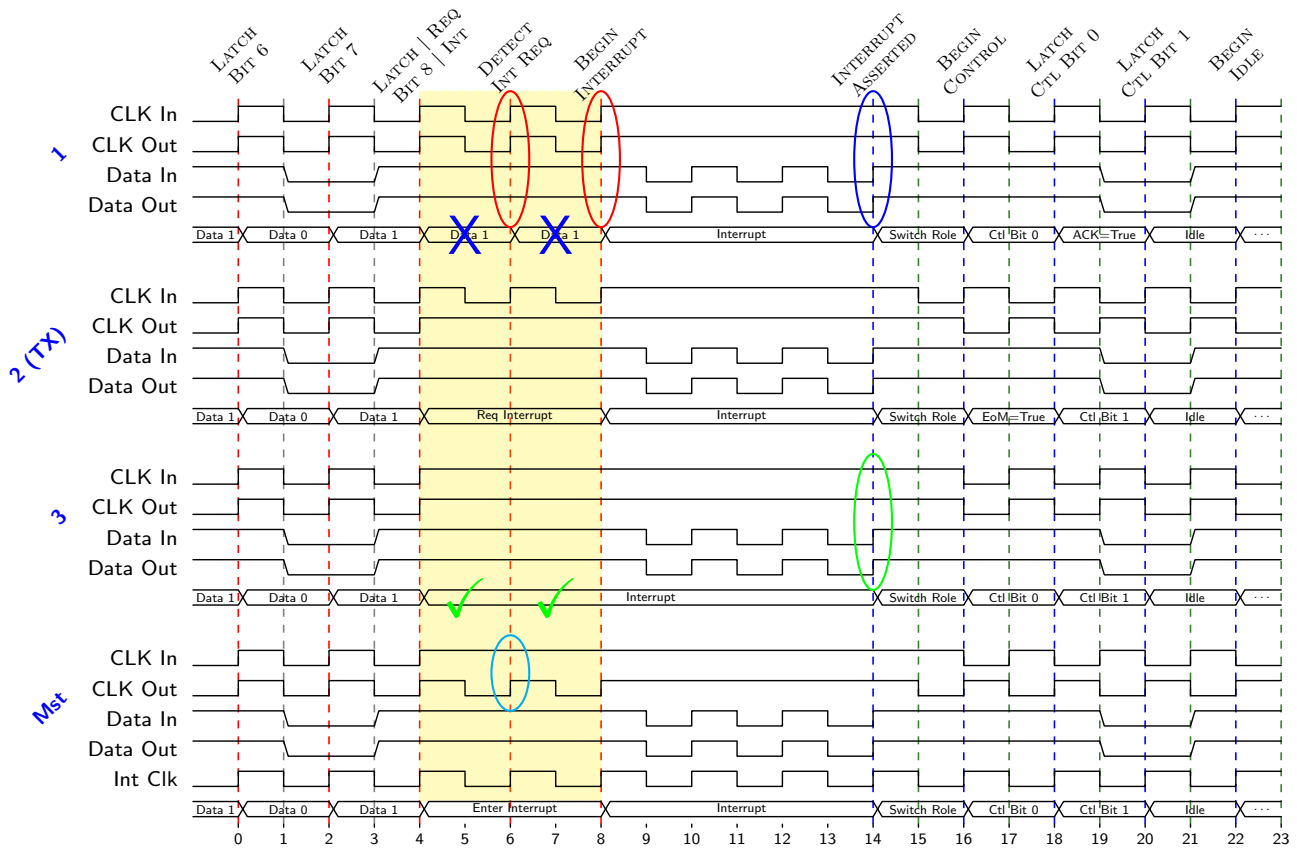


Figure 5: Detail of the Interrupt entry procedure and control bits. In this example, Node 2 is the TX node, thus when it elects to enter Interrupt it is already forwarding its data lines. Node 2 decides to enter Interrupt at time 4 by switching its CLK\_OUT mux from CLK\_IN to 1. At time 6, the master node detects that its CLK\_IN line never went low, and prepares to Interrupt the bus. At time 8 the master node stops driving clock edges and begins toggling the *data* line to interrupt the bus. After three data pulses, all nodes will enter Interrupt. Once the master node receives three pulses on DATA\_IN, it begins driving the clock again. Two control bits are sent, after which the bus returns to Idle.

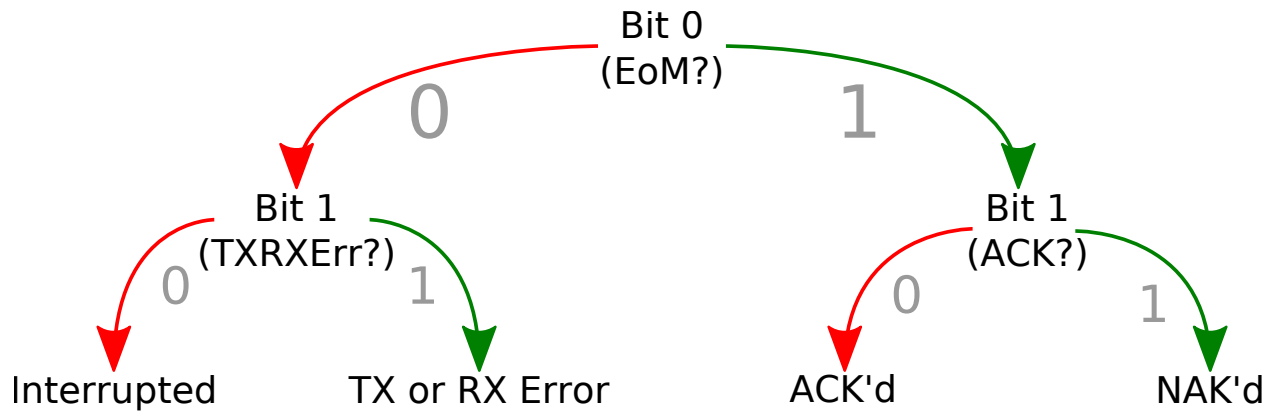


Figure 6: After an Interrupt, two control bits follow. The first bit is always set by the interrupter and indicates whether the Interrupt was to signal the end of message (EoM). If the EoM bit is high, the receiving node is responsible for driving the second bit to acknowledge the message. If the EoM bit is low, the interrupter is responsible for driving the second bit. The TX/RX Error (TRE) bit is set when a transmitting or receiving node encounters an error.

A MBus interrupt is defined as a series of pulses on the data line while the clock line remains high. After  $N$  pulses where  $N \geq 3$ , a node enters Interrupt<sup>10</sup>. Edges on the data line while the clock is low are ignored. This permits potentially racy changes of DOUT drivers to safely change on the falling edge of the clock without potentially introducing spurious interrupts.

The first rising clock edge after Interrupt is defined as “Begin Control”. The next two clock edges define the control bits.

### 3.4.1 Nesting Interrupts

In normal operation, interrupts are not permitted to “nest”, that is, the bus may not be interrupted while the control bits are being sent. If an Interrupt sequence occurs during an existing Interrupt (which may only occur if the bus is being “rescued” from some erroneous state), the new interrupt **must** present control bits 00. Any operations requested from the previous Interrupt state are aborted and the net Interrupt result is 00 (General Error).

## 3.5 Control Bits

The two bits after an Interrupt are defined as Control Bits. Figure 6 is a flowchart indicating the semantic meaning of the control bits and the resulting state. The first control bit is unconditionally driven by the interrupting node. If a transmitting node has interrupted to indicate the end of transmission, it will put a 1 on the bus for the first control bit. In all other cases the interrupter must drive 0 for the first control bit.

If the first control bit was 1 then the addressed receiver is responsible for driving the second control bit. If the transmission was sent to the broadcast address (7.1), all nodes—excluding the transmitter which forwards—should drive the second control bit low to acknowledge. The semantic provided to the transmitter of a broadcast message then is either (1) no nodes received the message or (0) *at least* one node received the message.

If instead the first control bit was 0 then the interrupter is responsible for driving the second control bit. If the interrupter is the transmitter or receiver **and** the issue is related to the current transmission (e.g. receiver buffer overflow) then the second bit should be set high. If the purpose of the interruption is unrelated to the current transmission (e.g. an external, high-priority, time-critical message) then the second bit should be set low.

<sup>10</sup> While two pulses is sufficient to distinguish an Interrupt from normal data transmission, three pulses provides protection against spurious entry to Interrupt from a glitch on data lines.



Unless the interruption carries a specific meaning as outlined in this section, the 00: INTERRUPTED state should be used for general or unclassified interruptions as it carries the least semantic meaning

### 3.6 Return to Idle

After latching the final Control Bit (time 20 in Figure 5), one final edge (time 22) is generated to formally enter IDLE. If, however, the data line is low at this edge, then it shall be considered the start of a new arbitration cycle instead. The master node will pull the clock low again in response and begin waiting for  $t_{long}$ .

By providing this edge, MBus enables member nodes with absolutely no sense of time the ability to receive, react to, and respond to messages (albeit with tight timing constraints).

*M3 Implementation Note:* (Referring to Figure 5) A broadcast sleep message cannot be considered a sleep until time 18 when the transmitter asserts that all the sent bits were desired to be sent. This leaves edges at time 20 and time 22 as the required two edges to power gate the layer.

This could be complicated, however, if a node elects (for some reason) to send a response to the broadcast sleep message. The sleep controller will still have four edges (Arbitration, Priority Drive, Priority Latch, Begin Transmission) to wake its bus controller, but the timing between arming its CLK\_IN fall detector and the master node pulling CLK\_IN low in response could cause the sleep controller to miss wakeup.

## 4 Power Design

The purpose of MBus is to support very low power operation. As such, it is expected that systems leveraging MBus may need to support power-gating all or part of the system. In this section, we discuss the requirements to support power-gated systems, how such systems integrate with MBus, and how power-oblivious chips can seamlessly interact with hyper power-conscious chips, promoting interoperability.

### 4.1 A Brief Background on Power-Gating

In the low-power design space, a simple and important concept is the ability to power-gate, or selectively disable, portions of a system that would otherwise be idle. By power-gating—removing power from that section of a chip—the power consumption of idle silicon goes to zero.

Several challenges with power-gating include: how to preserve state during idle windows, how to connect power-gated modules to other powered or power-gated modules, and how to wake and sleep power-gated modules deterministically. This document does not seek to address all these issues, rather to demonstrate how MBus can help system designers and the signals that are “safe” to utilize. For a more detailed reference power-gated design with MBus, consult the *MBus M3 Implementation Specification*.

In general, sleeping and waking power-gated modules requires a series of events to occur. In particular, the following signals define common power control design:

Signal Name	Function	Power-Up	Power-Down
POWER_ON	Controls Power-Gating	1st	2nd
RELEASE_CLK	Supply Clock to Internal Logic	2nd	2nd
RELEASE_ISO	Electrically Isolate Module I/O	3rd	1st
RELEASE_RST	(De)Assert Reset	4th	2nd

For the purposes of this document, we are concerned with two modules: (i) The block that interfaces with the bus itself—we define this as the **Bus Controller**—, and (ii) the node that is attached to the bus. With MBus, a completely power-gated node can seamlessly awaken its **Bus Controller** with no special assistance from the sending node or the master node. A **Bus Controller** can filter addresses, only waking the entire node for a message destined for that node. Additionally, a power-gated node with a simple always-on low power interrupt generator can exploit MBus features to generate the required edges with no specialization or externally synchronized knowledge of chip status. Finally, devices can reliably detect a shutdown message sent on the bus and use remaining control edges to shut down both the node and the node’s **Bus Controller**.

### 4.2 Waking the Bus Controller

Referring to edges from Figure 4, edges 1, 2, 3, and 4 provide the required signals. Mapping power edges to MBus protocol edges:

Arbitration	→	POWER_ON
Priority Drive	→	RELEASE_CLK
Priority Latch	→	RELEASE_ISO
Drive Bit 0	→	RELEASE_RST

In practice, most **Bus Controller** implementations will not require the **RELEASE\_CLK** signal as the MBus clock is (by definition) sufficient for all bus operations, however it is included in considerations for designs that may require it. A **Bus Controller** that is awoken using MBus edges will find its first rising clock edge to be Latch Bit 0, the MSB of the address, and should design state machines appropriately.

#### 4.2.1 Handling Interrupts During Wakeup

By specification, Interrupts are not permitted during arbitration. If an Interrupt occurred, an ignorant **Bus Controller** would hang, unable to make forward progress as the remaining edges would have been driven from clocking the control bits, causing the **Bus Controller** to interpret either Latch Control Bit 0 or Latch Control Bit 1 as the MSB of the destination address. After one or two more cycles, the bus would

become idle while the local **Bus Controller** waits indefinitely for more bits. This condition would eventually resolve itself if any other node elected to send a message but could not be resolved by any other means<sup>11</sup>.

As the **Bus Controller** was previously powered down, powering it down again before releasing isolation is by definition a null operation. As isolation is released on the final falling edge of arbitration and Interrupt may only occur while the clock is high, a wake-up that is not Interrupted is safe and canceling (re-power-gating) a wake-up that is Interrupted is safe. The challenge is then that the sleep controller module that generates the power control edges must also be capable of detecting an Interrupt. Whether this level of robustness is required is left as an implementation decision.

### 4.3 Waking the Node

If the **Bus Controller**'s address matches the destination address, it must wake whatever it is attached next up the chain, this means the clockless **Bus Controller** module must harvest clock edges from MBus to generate the power control signals.

One design point explicitly required by MBus is the acknowledgment of zero-length messages. Depending on application, a node may not require awakening for a zero-length message. Due to the nature of MBus Interrupt procedure, however, as many as two bits may be received that will be discarded (Figure 5). A **Bus Controller** design that attempts to minimize wakeups should therefore not begin the wakeup process until latching the 3rd data bit.

#### 4.3.1 Handling Interrupts During Wakeup

As the control bits provide ample edges, designers have more options for handling interrupted wakeup. In particular, the same argument regarding wakeup cancellation and arbitration from 4.2.1 applies: if isolation has not been removed, the node may simply be re-power-gated without issue.

A possibly simpler implementation can unconditionally complete the wakeup sequence while indicating that a transaction was started, but failed.

Ultimately, the important consideration is to draw attention to the fact that an Interrupt *may* occur during the **Bus Controller**'s issuing of wakeup signals (at any point) and a robust **Bus Controller** implementation must consider and handle the cases surrounding Interrupt.

### 4.4 Waking via Local Interrupt

It is also desirable for a clockless, power-gated MBus node to generate an interrupt and send a message<sup>12</sup>. As the node now wishes to send a message, it cannot use the arbitration edges as wakeup edges as it must be participating in the same arbitration to wake itself up. Bus-level solutions, such as extra initial wakeup edges, require a global sense of the current "power-state" of the bus as well as sacrificing the ability of power conscious sensors to be used with power oblivious designs.

Instead, power conscious nodes can exploit a robustness property of every MBus master node to generate wakeup edges. Section 6.13 **Failed Arbitration (Spurious Wakeup)** defines master node behavior in response to a glitch on the data line causing a spurious wakeup. By deliberately inducing such a glitch, a power-gated member node can generate enough pulses to wake itself up. Figure 7a shows an example waveform of an induced glitch, annotated with power control signals. Figure 7b demonstrates a simple inducer circuit.

Details on how the **Bus Controller** disambiguates between an RX-induced wakeup and an interrupt requested wakeup and other implementation issues are outside the scope of this document, but persons developing a power-gated system are highly encouraged to read the *MBus M3 Implementation Specification* as an example of how to design a power-gated system.

<sup>11</sup> Excepting things such as a local timeout and an external reset, but such a design is outside the scope of the discussion for a MBus member node.

<sup>12</sup> As example, consider an ultrasonic transducer, which can be used as a near zero-energy wakeup source by harvesting energy from an ultrasonic wakeup chirp.

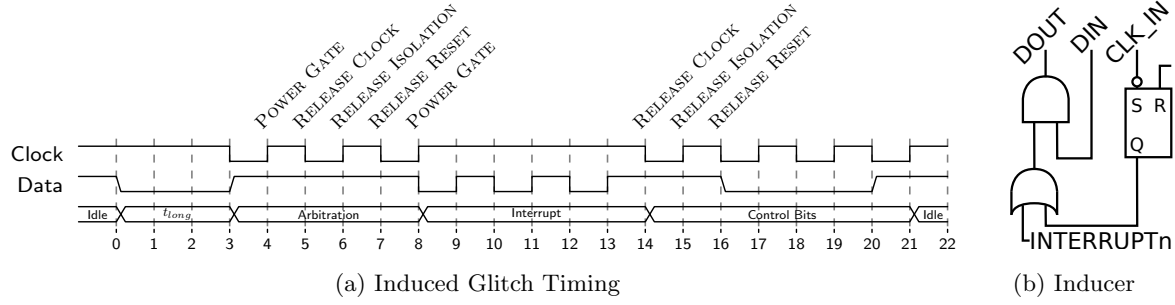


Figure 7: Induced Glitch. A node inducing a glitch uses the edge at time 3 to de-assert its bus request, instead electing to forward again. At time 4, the master node detects no winner of arbitration. The master node finishes the arbitration cycle as required by 6.13. The master node then immediately Interrupts the bus, indicating failure. An inducing node can harvest these clock edges to wake itself. The first set of power edges are used to wake the **Bus Controller**, the second set are used by the **Bus Controller** to wake the entire node. Figure 7b shows a simple glitch inducer circuit. The **INTERRUPTn** line is active low.

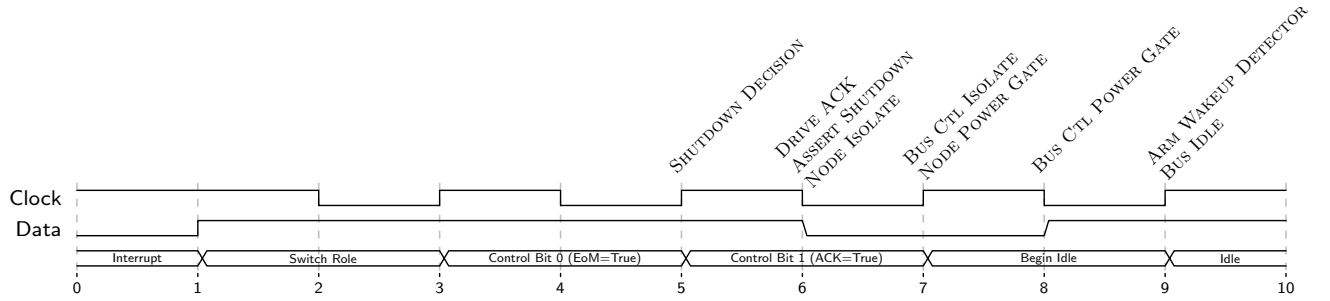


Figure 8: *Shutdown Timing*— The shutdown command is not confirmed until time 5 when the transmitter indicates a valid End of Message signal. At time 6, the bus controller acknowledges shutdown, asserts the **SHTDWN** signal to the sleep controller, and isolates the layer controller. At time 7, the sleep controller isolates the bus controller, and isolating the bus controller by definition power gates the layer controller. At time 8, the sleep controller power gates the bus controller, completing shutdown. At time 9 the bus is idle, and the sleep controller is waiting for the next wakeup.

## 4.5 Sleeping the Bus Controller, Node

Mbus edges can also be harvested to return both the node and the bus controller to sleep mode. While the exact message structure is left as implementation defined, Figure 8 demonstrates how the control edges following the End of Message bit may be used to put both the node and the **Bus Controller** to sleep.

Figure 8 delays the shutdown procedure until the transmitter asserts End of Message, indicating the shutdown message was not in error. In addition, Figure 8 has the advantage that a node transmitting a shutdown request can itself shut down with no further intervention or knowledge by the master node.

## 4.6 The Master Node, In Brief

Explicitly not discussed in this document is any indication of how a power-gated master node is implemented. The period  $t_{long}$  is permitted to be arbitrary in length with the express intent of allowing sleeping master nodes to wake themselves using the falling edge of **DIN** as a trigger. The actual means for waking the master node are outside the scope of this document however.

Similarly, sleeping a master node is outside the scope of Mbus functionality. A master node must sample its **DIN** line upon returning to Idle (time 9 in Figure 8) by the rules laid out in 3.6 Return to Idle. If the **DIN** line is high at that time after an implementation-defined shutdown message, the master node may safely enter sleep mode.

## 5 Addressing Design

One of the MBus design points is to serve as a system interconnect for a physically constrained system. With that in mind, MBus attempts to optimize for a system of complex connected components. In particular, MBus expects that an individual member node may itself be composed of multiple functional units, which may each be individually addressed.

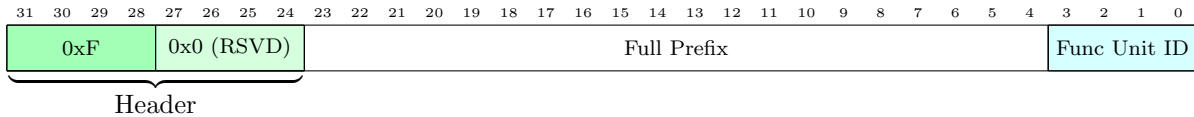
Supporting multiple, addressable functional units with only one set of exposed MBus signal pins could be done by instantiating several copies of a member node on each die, but this requires a bus interface module for every functional unit within an integrated chip. Instead, the bottom four bits of addresses are reserved to identify functional units. Nodes with more than sixteen functional units will require multiple addresses.

### 5.1 Address Types

MBus defines the term *prefix* to refer to the portion of the address that specifies which node is being addressed and reserves the term *address* to refer to a complete address—one that specifies a functional unit within a node.

Every MBus node has two prefixes, a *full* prefix and a *short* prefix. A short prefix is 4 bits long and a full prefix is 20 bits long. A short address is the composition of short prefix and a functional unit identifier. A full address is the composition of a header, a full prefix, and a functional unit identifier. To distinguish full and short addresses, the short prefix 0b1111 is reserved to identify a full address. The first four bits of the full address header is thus always 0b1111.

### 5.2 Full Prefix Assignment

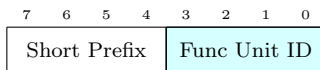


The purpose of full prefixes is to serve as a unique identifier for a node, akin to a product identifier. Full prefixes do not distinguish instantiations of a node, that is, multiple copies of a unique part will all have the same full prefix. This implies that multiple nodes in a single MBus instantiation may have the same full prefix. Bits 24-27 of the full address are reserved for other purposes (8.1.1 Resuming Interrupted Transfers). The remaining range, bits 4-23, are available to be utilized as full prefixes. If a node has more than 16 functional units, it may have multiple full prefixes. These full prefixes should be sequential.

- The full prefix 0x00000 is reserved as the broadcast prefix.

The full prefixes ranging 0x00000–0x0000F are reserved for legacy M3 devices. An allocation scheme for new full prefixes is currently undefined.

### 5.3 Short Prefix Assignment



The purpose of short prefixes is to uniquely identify nodes in an MBus system. Multiple nodes in a MBus instantiation **must not** have the same short prefix.

- The short prefix 0b0000 is reserved as the broadcast prefix.
- The short prefix 0b1111 is reserved to distinguish full addresses.

This leaves a remainder of 14 unique short prefixes. These short prefixes map to actual nodes instantiated in a MBus system. If there are multiple copies of the same node type (e.g. several external memory chips), each instance is given a unique short prefix. If a node has greater than 16 functional units it will be assigned multiple short prefixes, each mapping to one of the node's full prefixes.

In a MBus instantiation, short prefixes are assigned dynamically. Out of reset, all member nodes are assigned the short prefix **Unassigned Short Prefix: 0b1111**. After system power-on, some node shall send a series of **Enumerate Node** commands. MBus does not require that the master node send the **Enumerate Node**

commands, however any node containing the master node should be capable of enumerating bus members. Any node capable of performing enumeration **must** be capable of snooping another node's performing of enumeration and **must** update its local enumeration state from the snooped data.

Short prefixes should be assigned in ascending numerical order beginning with prefix 0b0010. It is the responsibility of the node(s) performing enumeration to ensure no duplicate short prefixes are assigned. As enumeration assignments are resolved by the topological priority arbitration, the short prefix assigned to a node should also correspond to its topological priority. This constraint may be violated in systems that do not enumerate prefixes in order, that use the **Invalidate Prefix** command to re-order short prefixes, or that use **Static Short Prefix Assignments** where the statically assigned prefixes do not match the topology. Short prefixes cannot be used to affect priority. Priority is determined exclusively by physical topology.

Short prefix assignments **must** be preserved when nodes are powered gated.

### 5.3.1 Static Short Prefix Assignments

All MBus nodes **must** support dynamic short prefix assignments. A node may have a default short prefix assignment. Upon the receipt of any **Enumerate Node** commands, a node with a default short prefix **must** immediately invalidate its default short prefix and participate in the enumeration process. A node with a default short prefix that receives an **Invalidate Prefix** command shall set its short prefix to **Unassigned Short Prefix: 0b1111**. A node with a default short prefix shall otherwise make no distinction from a dynamically assigned short prefix (e.g., it shall respond to a **Query Devices** command with its current short prefix, the static short prefix). **An allocation scheme for default short prefixes is not yet defined.**

## 6 Specifications

### 6.1 Granularity

Mbus sends data in units of 8 bit bytes. Transmissions must be modulo 8 bits in length, that is they must end on byte boundaries (for rationale, see [Granularity](#)).

### 6.2 Endianness: Byte-Level

Mbus sends from Byte 0...Byte N.

### 6.3 Endianness: Bit-Level

Mbus sends bytes most significant bit first (bit 7...0).

### 6.4 Minimum Message Length

The minimum legal message on Mbus is *zero* bits long. A zero length message can be used to query whether a device is present on the bus. A node receiving a zero length message addressed to it **must** ACK the message. Whether higher layers are indicated of a query message is implementation dependent.

See [6.12 Interrupt Rules](#) for details on the minimum uninterruptable message length.

### 6.5 Minimum Maximum Message Length

All Mbus master nodes must permit a minimum of 1 K (1024 bits) of data to be transmitted before aborting a transaction due to a hung transmitter.

The message length counter shall count the number of positive clock edges received after Begin Transmission at the master node's CLK\_IN port up to and including the last bit latched (Request Interrupt in Figure 5). The counter shall be an *inclusive* counter, that is the master node shall not Interrupt until it receives the  $N + 1$ th bit.

Commands to query and configure the upper bound are specified in [Broadcast Channel 2: Mbus Configuration](#).

### 6.6 Retransmission

There is no hardware re-transmission primitive. The hardware provides a transaction-level message acknowledgment, indicating whether the complete message was received or not. Retransmission of failed messages is left to software.

### 6.7 Minimum Buffer Size

All Mbus nodes are required to support both short and full addresses ([6.10 Addressing](#)). All Mbus nodes are further required to accept a minimum of 32 bits (4 bytes) of data in an individual transmission.

### 6.8 Buffer Overflow / Flow Control

On the average, the expectation in a Mbus system is that a transmitting node knows the capacity of the receiving node. If a receiving node does not have enough space for the current transmission, it **must** Interrupt the current transmission and indicate a receiver error (Control Bits: 01).

The receiver must Interrupt as soon as it is *certain* that it has exceeded its receive capacity. This detection must be performed carefully and must take into account that two bits beyond the complete transmission may be temporarily received (as is the case for Node 1 in Figure 5). The receiving state machine should not transition into DECIDED\_TO\_INTERRUPT until the 3rd bit of a byte that it does not have space for. The receiver may wait until the 8th bit to transition its state machine to request interrupt, but no later than that such that it is assured that it will interrupt the transmission before the transmitter attempts to interrupt to signal end of message even if the receiver is of lower priority.

## 6.9 Clock Speed

**TODO: What is appropriate specification for clock speeds?** I2C has defined ‘speed-classes’; SPI is a free-for-all; UART & co has bauds...

## 6.10 Addressing

MBus defines two types of addresses: *short* 8-bit addresses and *full* 32 bit addresses. No short address may begin with 1111 and all full addresses must begin with 1111. Address assignment is laid out in [5.3 Short Prefix Assignment](#) and [5.2 Full Prefix Assignment](#) respectively. Addresses 0x00, 0x01, and 0xffffffff are reserved. Their function is detailed in [7 Control and Configuration Messages](#).

## 6.11 Unassigned Short Prefix: 0b1111

Out of reset, nodes self-assign a short prefix of 0b1111. This short prefix shall be considered as a sentinel value where short prefixes are reported, indicating that the node does not currently have a short prefix assigned.

## 6.12 Interrupt Rules

MBus permits any node to interrupt any message for any reason. To allow for minimal forward progress, however, MBus requires that nodes permit a minimum of 32 bits (4 bytes) be transmitted without interruption. An exception is made for the transmitting node, which is permitted to send messages shorter than 32 bits.

Nodes wishing to interrupt must wait until either 41 or 65 clocks after Begin Transmission to interrupt the bus (depending on whether the current message was addressed to a short address or full address). The interrupting node must wait until it has latched the 33rd bit of data before attempting to interrupt<sup>13</sup>.

## 6.13 Failed Arbitration (Spurious Wakeup)

During normal arbitration, when the arbitration is resolved the master node’s DIN line should be low no matter who is participating. If the master node finds its DIN line is high on the arbitration edge, it would indicate that no one is participating. The master node must treat this as an error and reset the bus.

The master node **must not** immediately interrupt the bus, however. As laid out in [1.2 Power Gated Nodes](#), the arbitration edges may be used as input to node sleep controllers. To avoid potential issues related to half-waking power gated nodes, the master node must wait until Begin Transmission (that is, have generated four clock edges) before interrupting the bus.

The control bits **must** be driven by the master node. They **must** be 00, general error.

<sup>13</sup> If the node instead attempted to interrupt on the 32nd bit and was a topologically higher-priority node, it would override the transmitter during Interrupt priority arbitration, not permitting the transmitter to signal End of Message.



## 7 Control and Configuration Messages

Mbus reserves two prefixes: a *broadcast* prefix and an *extension* prefix. The broadcast prefix is used to command an control bus state on a clean channel, isolated from other users of the bus. The extension prefix is used in the short prefix space to identify full addresses. It is currently unused and reserved in the full prefix space.

### 7.1 Broadcast (Address 0x0X, 0xf000000X)

Mbus defines the broadcast short prefix as 0b0000 and the broadcast full prefix as 0x00000. Broadcast messages are permitted to be of arbitrary length. Messages longer than 32 bits may be silently dropped by nodes with small buffers. A node **must not** interrupt a broadcast message to indicate buffer overflow. Other interrupts are permitted for broadcast messages greater than 4 bytes in length.

As broadcast messages by definition target all endpoints, assigning functional unit ID targets does not make sense. Instead, the functional unit ID field is used to define broadcast *channels*. Broadcast channel selection is used to differentiate between the different types of broadcast messages. Mbus reserves half of these channels and leaves the rest as implementation defined. The MSB of the broadcast channel identifier (address bit 3) shall identify Mbus broadcast operations. If the MSB is 0 it indicates an official Mbus broadcast message as specified in this document and subsequent revisions. Broadcast messages with a channel MSB of 1 are implementation defined. It is recommended that nodes leveraging implementation-defined broadcast channels provide a mechanism to dynamically select broadcast channel to help mitigate conflicts.

A broadcast message that is not understood **must** be completely ignored. During acknowledgment, an ignorant node shall forward.

#### 7.1.1 Broadcast Messages and Power-Gating

As power-gating of nodes is a Mbus design consideration, Mbus also defines the responsibilities of power-gated nodes for broadcast messages. The **Bus Controller** **must** wake for every message. Wakeup shall occur during the arbitration phase of a message. For non-broadcast messages, the **Bus Controller** decides whether it will wake the node once it has received the entire address. To simplify **Bus Controller** design, the same constraint is imposed on broadcast messages: the **Bus Controller** should be able to decide whether it will wake the node after receiving the complete address. The implication is that the rules for waking nodes in response to broadcast messages are grouped by broadcast channel. This requirement **must** hold true for all Mbus broadcast channels and should hold true for any implementation-defined channels.

Some messages (e.g. **Query Devices**) require a response but may not require waking the node. The message semantics take priority over the channel power rules. That is, if a message must be responded to and the **Bus Controller** logic implementation is too simple to respond, the **Bus Controller** **must** wake the node to respond.

The Mbus broadcast channel power rules are summarized here:

- **Broadcast Channel 0: Node Discovery and Enumeration**
  - Nodes responsible for performing enumeration **must** be woken.
  - All other nodes may be woken.
- **Broadcast Channel 1: Power**
  - Nodes **must** obey the indicated power request.
  - Nodes interested in snooping not affected by the request may be woken.
- **Broadcast Channel 2: Mbus Configuration**
  - The control node **must** wake to handle the request.
  - If a node's behavior would change as a result of the request, it **must** wake and handle the request.
  - Nodes interested in snooping not affected by the request may be woken.
- **Broadcast Channels 4-6: Reserved**
  - Nodes capable of receiving arbitrary messages (e.g. CPU) may be woken.
  - Fixed-function nodes (no software) **must not** be woken.
- **Broadcast Channel 7: Data**

- Nodes that ignore the Data channel **must not** be woken.
- All other nodes may be woken.

In all cases where a node may wake up, nodes shall be assumed to not wake up unless explicitly stated otherwise.

### 7.1.2 Broadcast Channels and Messages

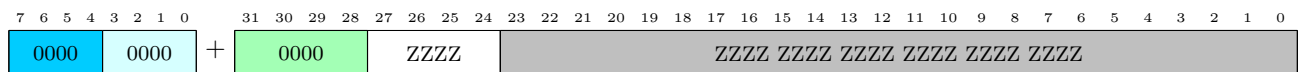
This section breaks down all of the defined MBus broadcast channels and messages. All undefined channels are reserved and shall not be used. A node receiving a broadcast message for a reserved channel shall ignore the message. It **must not** acknowledge a message on a reserved channel and **must** forward during the acknowledgment cycle.

All MBus broadcast messages, except those sent on **Broadcast Channel 7: Data**, follow a common template. The messages are 32 bits long. The four most significant bits identify the message type/command. Some messages do not require all 32 bits. The unused bits are named *insignificant bits*. Messages may be truncated, omitting the insignificant bits on the wire<sup>14</sup>.

All examples are shown with short addresses for space. There is no distinction between the use of the short or full broadcast address. Bitfields are presented **Address + Data**. Addresses are broken down into the **Broadcast Prefix** and the **Broadcast Channel**. Data is broken down into a **Message Type Specifier** and the message itself. In the bitfields, 0 and 1 indicate bits that must be set to that value, X indicates bits that depend on the current message, and Z indicates bits that should be *ignored*—accept any value, send as 0. **Insignificant Bits** are also indicated as Z.

**Broadcast Channel 0: Node Discovery and Enumeration** Channel 0 is used for messages related to node discovery and enumeration. Channel 0 messages either require a response or are a response. Channel 0 response messages should not be sent unless solicited. Nodes that are capable of performing enumeration **must** snoop all enumeration related messages (**Enumerate Node**, **Query/Enumerate Response**). If a node's enumeration controller is power-gated, it must be woken for these messages.

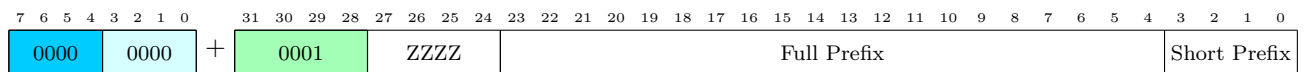
#### Query Devices



The query devices command is a request for all devices to broadcast their static full prefix and currently assigned short prefix on the bus. Every MBus node must prepare a **Query/Enumerate Response** when this message is received.

*All nodes are required to support this message and respond.*

#### Query/Enumerate Response



This message is sent in response to a **Query Devices** or **Invalidate Prefix** request. When responding to **Query Devices**, every node will be transmitting their address, and nodes should anticipate losing arbitration several times before they are able to send their response.

The top four bits of the data field identify the message as a Query Response. The next four bits are ignored. The following 20 bits contain the full prefix of the node. The final 4 bits are the currently assigned short prefix. Nodes that have not been enumerated should report a short prefix of 0b1111.

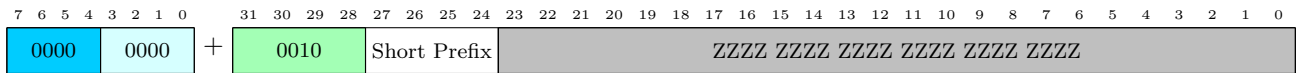
This message must be sent in response to **Query Devices** or **Enumerate Node**. When responding to **Query Devices**, nodes **must** retry until the message is sent. When responding to **Enumerate Node**, nodes **must not** retry sending if arbitration is lost and **must** retry sending if interrupted<sup>15</sup>.

<sup>14</sup> With the caveat that all MBus messages must be byte-aligned. Some insignificant bits may still be sent on the wire as a consequence.

<sup>15</sup> An interrupt should not occur during this message. Such an interrupt would be an error.

All nodes are required to support this message.

### Enumerate Node

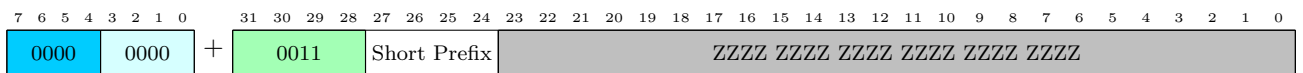


This message assigns a short prefix to a device. All nodes that receive this message and do not have an assigned short prefix **must** attempt to reply with a **Query/Enumerate Response**. Nodes shall perform exactly one attempt to reply to this message. The node that wins arbitration shall be assigned the short prefix from this message. Nodes that lose arbitration shall remain unchanged.

Nodes that have an assigned short prefix shall ignore this message.

All nodes are required to support this message and respond if appropriate.

### Invalidate Prefix

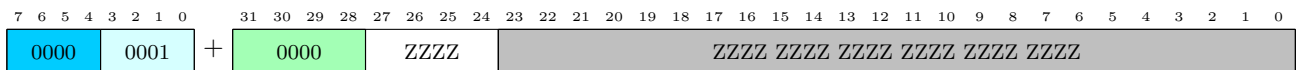


This message clears the assignment of a short prefix. The bottom 4 bits specify the node whose prefix shall be reset. A node shall reset its prefix to **Unassigned Short Prefix: 0b1111**. If the prefix to clear is set to **Unassigned Short Prefix: 0b1111**, then all nodes shall reset their prefixes.

**Broadcast Channel 1: Power** Channel 1 is used to query and command the power state of MBus nodes. Power-oblivious nodes may ignore channel 1. Power-aware nodes whose power model does not align well with these commands may ignore channel 1 messages *except All Sleep*. All nodes capable of entering a low-power state **must** enter its lowest power state in response to an **All Sleep** message.

Nodes are implicitly waked when a message is addressed to them, explicitly waking a node is unnecessary to communicate with it. The definition of “wake” and “sleep” are implementation-defined. Nodes interested in snooping power status may be woken for channel 1 messages, but care should be taken to ensure that a node does not wake itself up to snoop in the process of instructing itself to sleep.

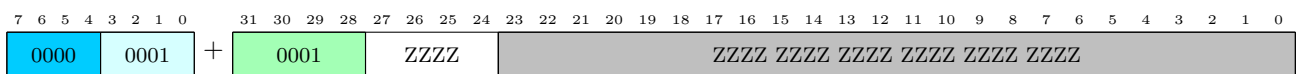
### All Sleep



All nodes receiving this message **must** immediately enter their lowest possible power state. The bottom 28 bits of this message are reserved and should be *ignored*.

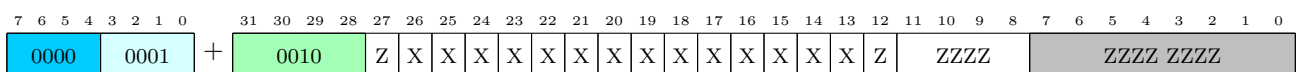
All power-aware nodes are required to support this message.

### All Wake



All nodes receiving this message **must** immediately wake up. The bottom 28 bits of this message are reserved and should be *ignored*.

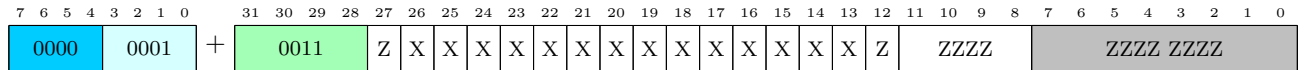
### Selective Sleep By Short Prefix



This message instructs selected nodes to sleep. The 16 bits of data are treated as a bit vector, mapping short prefixes to bit indicies. That is, the node with short prefix 0b1101 is controlled by the second bit

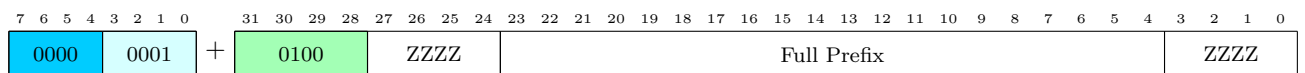
received (bit 18 in the bit vector above). If a bit is set to 1, the selected node **must** enter sleep mode. If a bit is set to 0, the selected node should not change power state. The bits for prefixes 0b1111 and 0b0000 are ignored.

#### Selective Wake By Short Prefix



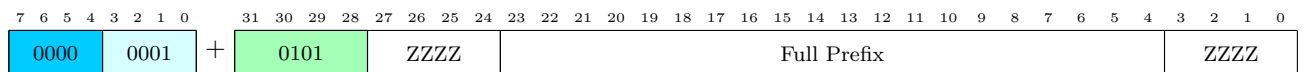
This message instructs selected nodes to wake. The 16 bits of data are treated as a bit vector, mapping short prefixes to bit indices. That is, the node with short prefix 0b1101 is controlled by the second bit received (bit 18 in the bit vector above). If a bit is set to 1, the selected node **must** wake up. If a bit is set to 0, the selected node should not change power state. The bits for prefixes 0b1111 and 0b0000 are ignored.

#### Selective Sleep By Full Prefix



This message instructs selected nodes to sleep. Any node whose full prefix matches **must** enter sleep.

#### Selective Wake By Full Prefix



This message instructs selected nodes to wake. Any node whose full prefix matches **must** wake up.

**Broadcast Channel 2: MBus Configuration** The purpose of this channel is to configure any MBus parameters. Commands issued on channel 2 **must** be targeted for the control node. By utilizing a broadcast channel all interested nodes can easily “subscribe” to configuration messages. Using a broadcast channel also permits nodes to hard-code the address for configuration messages.

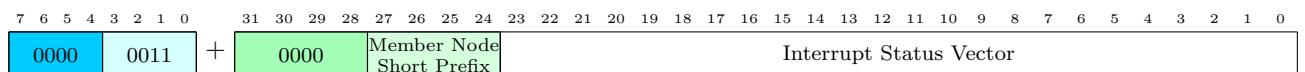
The current MBus specification does not define any channel 2 messages. Possible future messages include:

- Maximum message length
- Clock speed
- $t_{long}$  value?
- ...?

**Broadcast Channel 3: Member Node Events** The purpose of this channel is for member nodes to announce information to multiple interested parties. It also permits the simplification of a member node design by permitting the member node to hard-code a destination address for certain classes of messages (e.g. interrupts).

Member node event messages **must** also include a **Member Node Identifier**—the node’s current short prefix—to disambiguate member node events. It is possible, and acceptable, for a member node to generate a member node event message before enumeration has completed. In this case, a member node shall send its current short prefix: **Unsigned Short Prefix: 0b1111**.

#### Member Node Level Interrupt



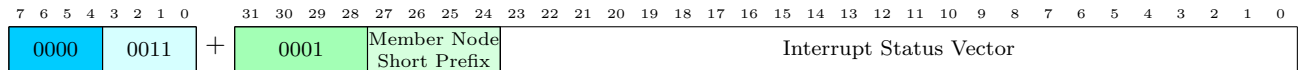
This message announces that an interrupt has occurred on this member node. The semantics of a MBus level interrupt dictate that an interrupt must be explicitly cleared. A member node **must not** generate another level interrupt message *for the same interrupt* until that interrupt is explicitly cleared. A *different* interrupt may occur before the first interrupt is cleared, in which case the bit vector **must** indicate both

interrupts as active. Member nodes are permitted to batch or delay interrupts if appropriate, that is, a single level interrupt message may indicate the multiple interrupts have occurred.

Clearing interrupts is **not** a Channel 3 message, as doing so would require all member nodes to wake for every Channel 3 message. As a consequence, systems with multiple nodes capable of clearing a member node interrupt should develop another means of coordinating the responsibility. A member node designed to have multiple potential controlling nodes should consider defining a broadcast control channel.

MBus defines active to be logical 1. Member nodes whose internal interrupts are active low must invert the signal before broadcasting the interrupt vector. The remaining 24 bits of this message are defined as a bit vector representing the current status of 24 interrupts.

#### Member Node Edge Interrupt



This message announces that an interrupt has occurred on this member node. The semantics of a MBus edge interrupt dictate that the interrupt is implicitly cleared. That is, after the broadcast message announcing the interrupt has been sent the same interrupt is eligible to fire again. Member nodes are permitted to batch or delay interrupts if appropriate, that is, a single edge interrupt message may indicate multiple interrupts have occurred.

MBus defines active to be logical 1. Member nodes whose internal interrupts are active low must invert the signal before broadcasting the interrupt vector. The remaining 24 bits of this message are defined as a bit vector representing the current status of 24 interrupts.

#### Broadcast Channels 4-6: Reserved

**Broadcast Channel 7: Data** Channel 7 is used to send arbitrary data to every addressable entity on the bus. MBus places no further restrictions or structure on channel 7 messages. Channel 7 is intended for use by more flexible software nodes, though any node may listen or transmit on channel 7.

## 7.2 Extension (Address 0xffffffff)

This address is reserved for future extensions to MBus.

## 8 ToDo

### 8.1 Future Extensions

There are properties that would be nice to have, but introduce greater complexity. While they are not in the current design, some methods for designing them are considered here such that they are still feasible for future implementation in a backwards-compatible manner.

#### 8.1.1 Resuming Interrupted Transfers

Consider a separate ‘bulk-transfer’ address such that nodes which support resumable transfers have two addresses (and a corresponding broadcast query for such addresses).

A bulk transfer always begins with the sending node’s full address both as unique token and so the bulk protocol can respond if needed. Transfers begin with both ends’ counters synchronized at 0 bits. If an interruption occurs, the RX node saves as much as it has RX’d thus far. After the bus is available again the RX node is responsible for sending the TX node a message composed of RX\_address+bits\_thus\_far. If the RX node does not send such a message, the TX node may solicit one; if the RX node has dropped the suspended transaction for any reason, it simply responds 0 bits.

Nodes that wish to permit multiple simultaneous bulk transfers could expose several bulk addresses. If an A→B transfer is interrupted and C attempts a C→B transfer, B would indicate busy by interrupting immediately after the bulk address was received with a 01 RX Error Interrupt sequence.

**Full addresses not unique?** Probably fair to make the assertion that only one functional unit per node can accept bulk transfers. Repurpose the functional unit id field to contain the short prefix?

## 9 Document Revision History

- Revision 0.2 (rXXXX) – May 5, 2013
  - Add Broadcast Channel 3
  - Clean up some of the more dated requirements text
  - Terminology change, “control” to “master” node
- Revision 0.1 (r4333) – Apr 22, 2012
  - Initial revision