MBus Specification

<*mbus-team@umich.edu*>
Pat Pannuto, Yoonmyung Lee, Ye-Sheng Kuo, ZhiYoong Foo, Ben Kempke, David Blaauw, and Prabal Dutta

Revision 1.0-alpha

Last Updated: June 14, 2015

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 only push-pull 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 clockless operation. MBus requires one more capable node to act as a bus *mediator* node, whose primary duties are providing the MBus clock and mediating arbitration.

The authoritative home of this specification is `http://mbus.io`. Updates and errata shall be posted there.

Contents

1 Node Design

The MBus defines two *physical* types of nodes: member nodes and a mediator node. An instantiation of MBus must have one and only one mediator node and must have at least one member node (N≥1). The maximum number of member nodes is a function of clock speed[1].

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≥0) of forwarding nodes are permitted.

1.1 Physical Design

The physical design of a member and mediator node is the same, each must expose a `DIN`, `DOUT`, `CLKIN` and `CLKOUT` pad.

TODO: Add documentation of EIO Debug pads

1.1.1 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[2] 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[3].

1.1.2 Mediator Node

A mediator node requires 4 signals:

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

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

1.1.3 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).

[1] 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 nodes * 20 ns. MBus designers are obligated to consider all possibly delays and thus define a maximum feasible clock speed, however, for many designs the maximum clock speed theoretically possible will likely exceed the plausible speed for the given power budget.

[2] 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.

[3] However, the only strict requirement is the 10 ns propagation delay, thus careful designers may violate this requirement if necessary.

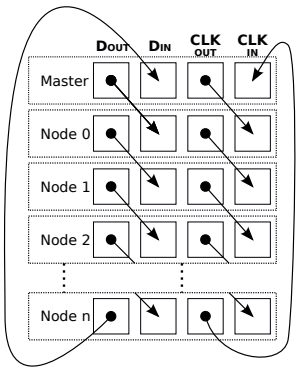


Figure 1: MBus Physical Topology. High-level picture of MBus physical design. Member nodes and a mediator node are connected in a loop, with data and clock lines forming independent rings.

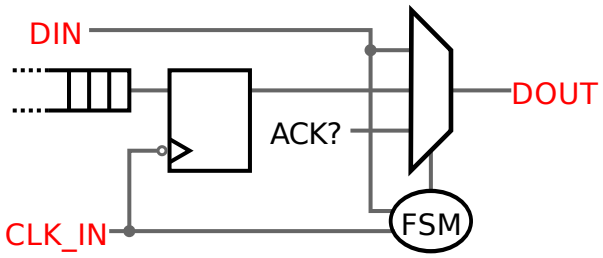


Figure 2: Logical Model. The Finite State Machine selects between the three modes a node can be in. Top: *forwarding*, Mid: *transmitting*, Bot: *acknowledging*. This model omits some of the subtleties of arbitration, see 1.2.2 Transmitting.ARBITRATE for details.

  - 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 2.2 for more 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.

  An MBus instance must have a minimum of two nodes (one mediator 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.

1.1.4 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.

TODO: Update with EIO Debug protocol

1.2 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 mediator node adopts these same three personalities, but its behavior differs slightly during arbitration (2.2).

1.2.1 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`.

Mediator Node Exception: In IDLE, the mediator node does not forward `DIN` to `DOUT`.

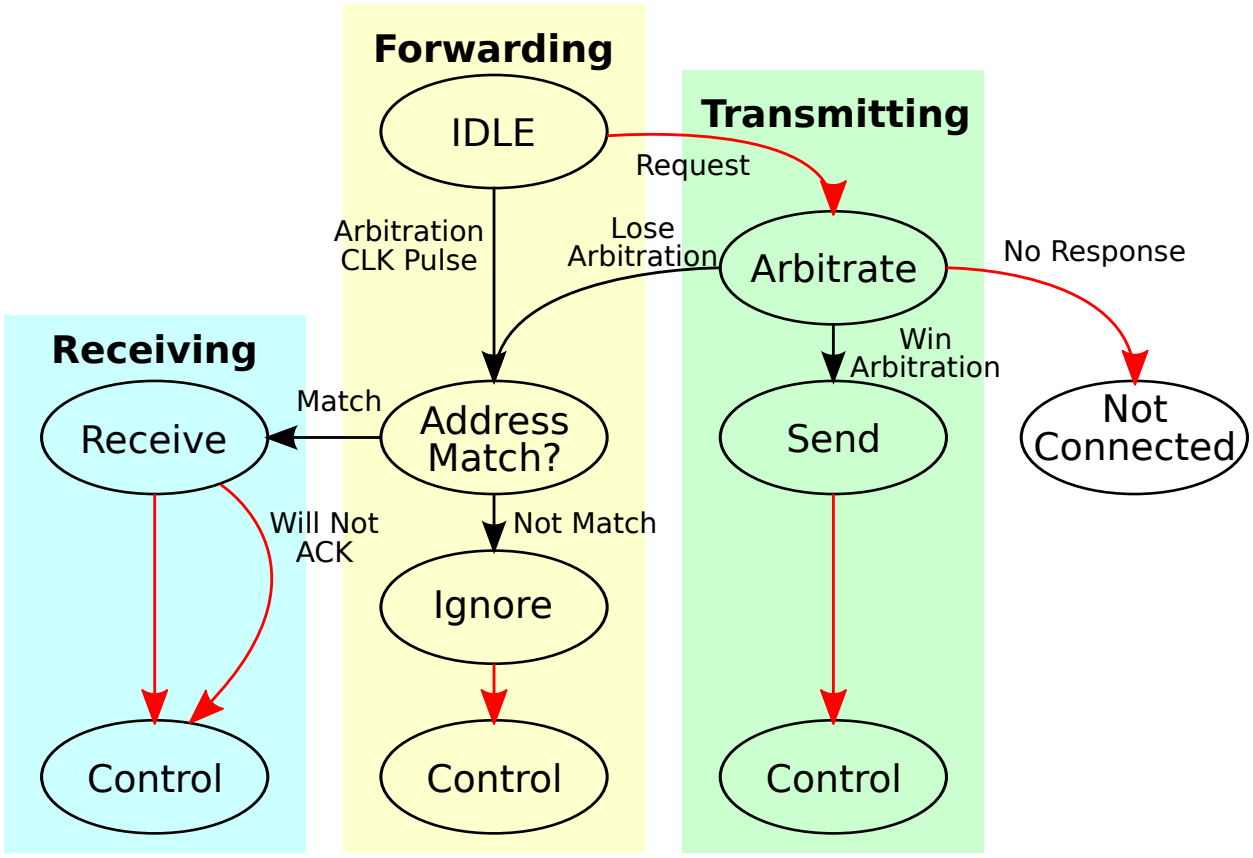


Figure 3: FSM describing the high-level logical behavior of MBus nodes. Black arrows indicate transitions that occur on bus clock edges. Not shown are implicit arrows from any state to Forwarding.CONTROL. From any CONTROL state, nodes progress to IDLE.

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 an interjection. Power-conscious designs may safely power gate everything except interjection detector circuitry while in IGNORE.

1.2.2 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[4].

The ARBITRATE state is left when the CLK line is pulled high by the mediator 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 to

[4]To envision the case defended against here, picture a tall stack of nodes, where the bottom node requests the bus. The mediator node pulls CLK low in response. Shortly before the mediator 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.

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[5]—the node should consider itself as disconnected.

Details of priority arbitration are omitted here for simplicity. See 2.2 Arbitration for details.

Mediator Node Exception: The mediator node always wins arbitration. Note that when this occurs, the mediator node's `DIN` will be low.

Mediator Node Exception: If a mediator 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 2.3 Message Transmission.

A transmitting node completes its transmission by interjecting (2.4 Interjection) 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.

1.2.3 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 an interjection.

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 interject the bus itself. A receiver will interject a transmission to indicate that its RX buffer has been overrun and the current transmission must be aborted.

1.2.4 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 mediator 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.

[5] TODO: TBD

2 Bus Design

During normal operation, MBus remains in Bus Idle (2.1). A transmission begins with Arbitration (2.2), then Message Transmission (2.3). The transmitter then Interjects (2.4) the bus and indicates the complete message was sent. During the Control (2.5) period acknowledgment is negotiated. After Control, MBus returns to Idle or may optionally send a response message.

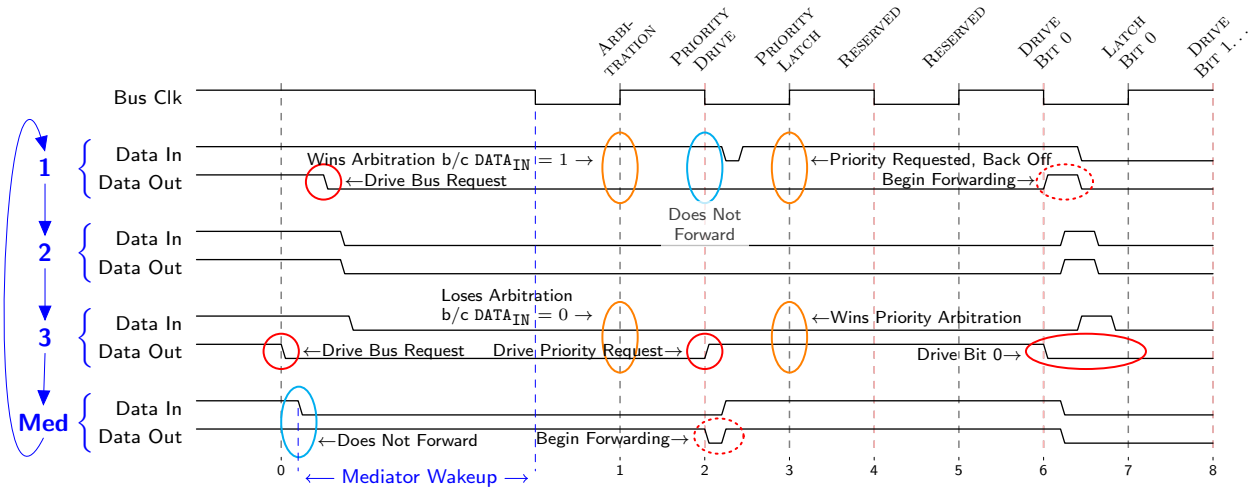


Figure 4: MBus Arbitration. To begin a transaction, one or more nodes pull down on DOUT. Here we show node 1 and node 3 requesting the bus at nearly the same time (node 1 shortly after node 3). Node 1 initially wins arbitration, but node 3 uses the priority arbitration cycle to claim the bus. The propagation delay of the data line between nodes is exaggerated to show the shoot-through nature of MBus. Momentary glitches caused by nodes transitioning from driving to forwarding are resolved before the next rising clock edge.

2.1 Bus Idle

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

2.2 Arbitration

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

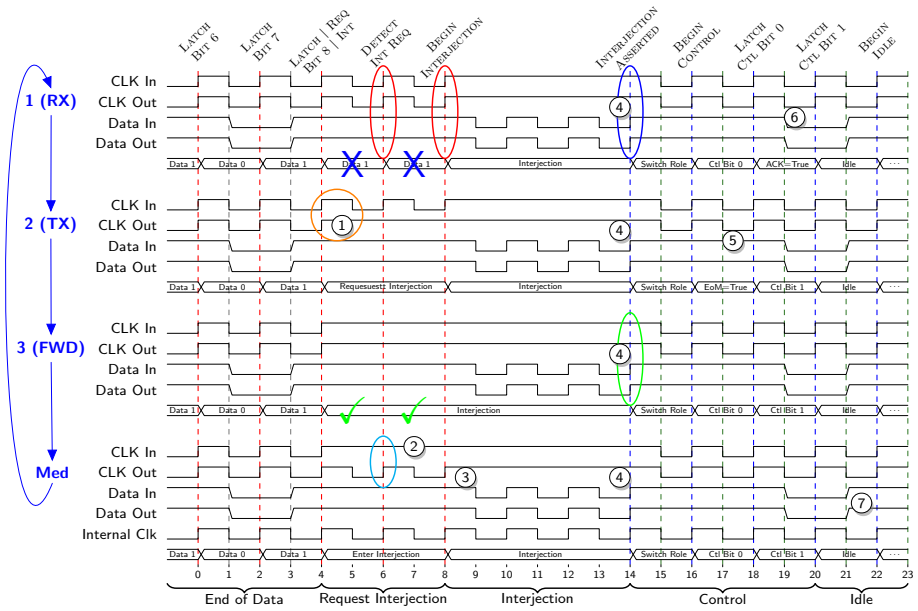
To request to transmit on the bus, a node should pull its DOUT line low. All member nodes remain in forwarding state during arbitration, thus when their DIN line goes low, they are obligated to pull their DOUT line low. The mediator node, however, does **NOT** pull its DOUT line low in response to its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



(a) Interjection timing with successful TX and ACK.

1. After the transmitter sends all of its data it requests interjection by not forwarding CLK.
2. The mediator detects that a node has stopped forwarding CLK.
3. The mediator stops toggling CLK and begins toggling DATA – the interjection sequence.
4. After interjection, the mediator begins clocking again. Node 1 discards the extra bits because they are not byte-aligned.
5. The transmitter signals a complete message by driving Control Bit 0 high.
6. The receiver ACK's the message by driving Control Bit 1 low.
7. After control, the mediator stops forwarding DATA, driving it high, and returning the bus to idle.

(b) Interjection Events.

Figure 5: MBus Interjection and Control. The MBus interjection sequence provides a reliable in-band reset signal. Any node may request that the mediator interject the bus by holding $\texttt{CLK}_{\texttt{OUT}}$ high. The mediator detects this and generates an interjection by toggling DATA while holding CLK high. An interjection is always followed by a two-cycle control sequence that defines why the interjection occurred.

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 mediator node in the data loop. That is, the furthest node from the mediator node, the node whose DIN is connected to the mediator node's DOUT, has the highest priority of member nodes. The closest node to the mediator node, the node whose DOUT is connected to the mediator node's DIN, has the lowest priority.

At the end of t_{long}, the mediator 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.

2.3 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.

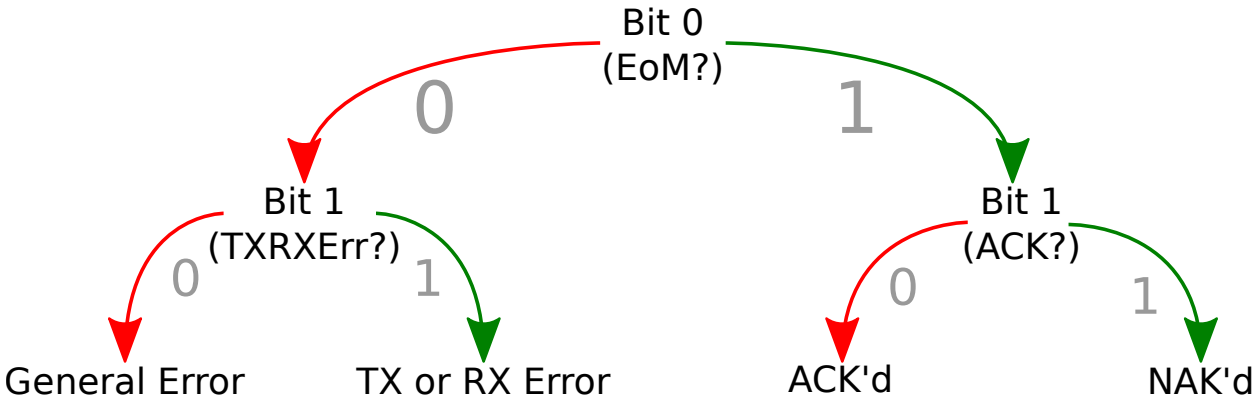


Figure 6: Control Bits. After an interjection, two control bits follow. The first bit is always set by the interjector and indicates whether the interjection was to signal the end of message (EoM). If the EoM bit is high, the receiving node is responsible for driving the second bit to acknowledge the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

- 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 8.10 Addressing). Once a transmitter has sent the complete transmission it interjects the bus.

Legal transmissions on MBus must be byte-aligned (8.1 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).

2.4 Interjection

During normal operation, interjection is only permitted after the first 32 bits of data have been transmitted (8.12 Interjection Rules). The interjection mechanism, however, is also the MBus reset mechanism, and as such member node interjection detectors **must** always be active whenever a node is not Idle.

A MBus interjection 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 interjection[6]. 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 interjections.

The first rising clock edge after interjection is defined as "Begin Control". The next two clock edges define the control bits.

2.4.1 Nesting Interjections

In normal operation, interjections are not permitted to "nest", that is, the bus may not be interjected while the control bits are being sent. If an interjection sequence occurs during an existing interjections (which may only occur if the bus is being "rescued" from some erroneous state), the new interjection **must** present control bits `00`.

2.5 Control Bits

The two bits after an interjection 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

[6] While two pulses is sufficient to distinguish an interjection from normal data transmission, three pulses provides protection against spurious entry to interjection from a glitch on data lines.

interjecting node. If a transmitting node has interjected to indicate the end of transmission, it will put a `1` on the bus for the first control bit. In all other cases the interjector 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 (5.1), 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 interjector is responsible for driving the second control bit. If the interjector 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 interjection is unrelated to the current transmission (e.g. an external, high-priority, time-critical message) then the second bit should be set low.

Unless the interjection carries a specific meaning as outlined in this section, the 00: General Error state should be used for general or unclassified interjections as it carriers the least semantic meaning.

2.6 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 mediator 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).

3 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.

3.1 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, each MBus node has two modules: (i) The block that interfaces with the bus itself—we define this as the `Bus Controller`—, and (ii) the block that comprises the rest of the node—we define this as the `Layer`. With MBus, a completely power-gated node can seamlessly awaken its `Bus Controller` with no special assistance from the sending node or the mediator node. A `Bus Controller` can filter addresses, only waking the `Layer` for a message destined for that node.

Additionally, a power-gated node with a simple always-on low power interjection 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 `Layer` and the `Bus Controller`.

3.2 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.

3.2.1 Handling an Interjection During Wakeup

By specification, interjection is not permitted during arbitration. If an interjection 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

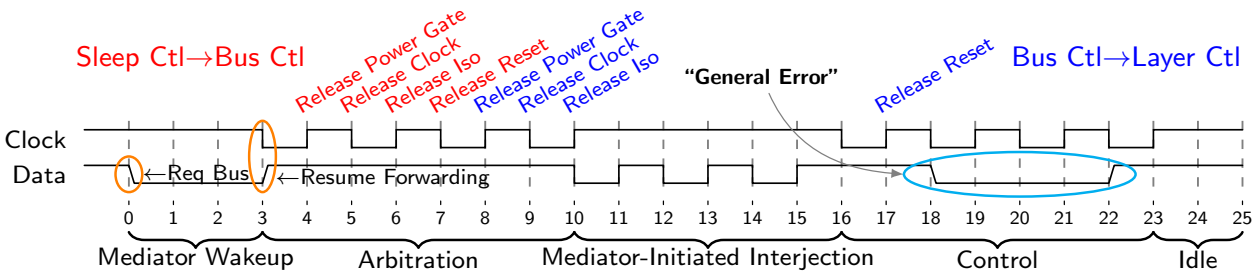


Figure 7: MBus Wakeup. Power-gated nodes repurpose the arbitration edges to wake the bus controller before data transmission starts. To self-wake, nodes can initiate a null transaction (shown here) by pulling down `DATA` and then resuming forwarding `DATA` before the arbitration edge. If no other node arbitrates, the mediator will detect no winner, raise a general error, and return the bus to idle. Arbitration produces enough edges to wake bus controllers before addressing. The null transaction produces enough edges to wake all of the MBus hierarchical power domains in a manner that is transparent to non-power-aware devices.

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.[7]

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 an interjection may only occur while the clock is high, a wake-up that is not interjected is safe and canceling (re-power-gating) a wake-up that is interjected is safe. The challenge is then that the sleep controller module that generates the power control edges must also be capable of detecting an interjection. Whether this level of robustness is required is left as an implementation decision.

3.3 Waking the Layer

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 the MBus interjection 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.

3.3.1 Handling an Interjection During Wakeup

As the control bits provide ample edges, designers have more options for handling an interjection during wakeup. In particular, the same argument regarding wakeup cancellation and arbitration from 3.2.1 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 interjection *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 interjection during `Layer` wakeup.

3.4 Waking via Induced Glitch

Section 8.13 Failed Arbitration (Spurious Wakeup) defines mediator 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

[7] 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.

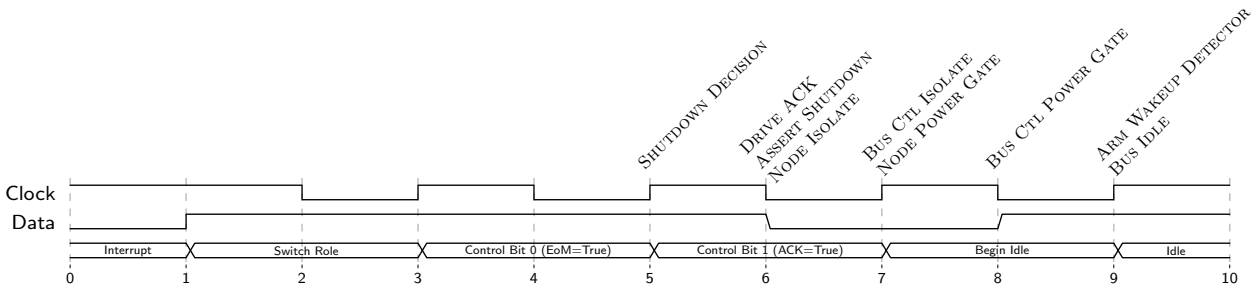


Figure 8: Shutdown Timing. The shutdown command is not confirmed until time 5 when the transmitter indicates a valid End of Message signal. At time 6, the bus controller acknowledges shutdown, asserts the `SHTDWN` signal to the sleep controller, and isolates the layer controller. At time 7, the sleep 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.

node can generate enough pulses to wake itself up. Figure 7 shows an example waveform of an induced glitch, annotated with power control signals.

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.

3.5 Sleeping the `Bus Controller`, `Layer`

MBus edges can also be harvested to return both the `Layer` and the `Bus Controller` to sleep mode. 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 mediator node.

3.6 The Mediator Node, In Brief

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

Sleeping a mediator node is also outside the scope of MBus functionality. A mediator node must sample its `DIN` line upon returning to Idle (time 9 in Figure 8) by the rules laid out in 2.6 Return to Idle. If the `DIN` line is high at that time after an implementation-defined shutdown message, the mediator node may enter sleep.

4 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.

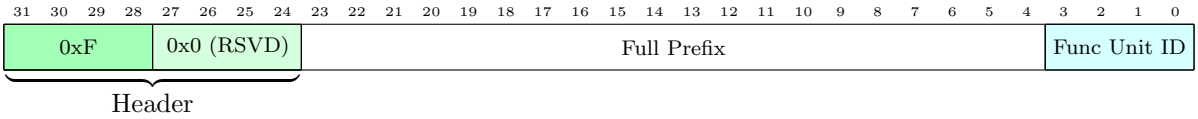
Supporting multiple, addressable functional units with only one set of exposed MBus signal pins could be done by instantiating several copies of a member node on each die, but this requires a bus interface module for every functional unit within an integrated chip. Instead, the bottom four bits of addresses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

4.1 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`.

4.2 Full Prefix Assignment

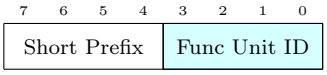


The purpose of full prefixes is to serve as a unique identifier for a node, akin to a product identifier. Full prefixes do not distinguish instantiations of a node, that is, multiple copies of a unique part will all have the same full prefix. This implies that multiple nodes in a single MBus instantiation may have the same full prefix. Bits 24-27 of the full address are reserved for other purposes (**?? ??**). 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.

4.3 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 mediator node send the Enumerate Node commands, however any node containing the mediator 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 a Static Short Prefix Assignment 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.

4.3.1 Static Short Prefix Assignment

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.

5 MBus Protocol Design

To maximize device interoperability, MBus defines a higher-level protocol for basic point-to-point register and memory access. MBus also defines a protocol for broadcast messages.

MBus reserves two prefixes: a *broadcast* prefix and an *extension* prefix. 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.

5.1 Broadcast Messages (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** interject a broadcast message to indicate buffer overflow. Interjections are permitted for broadcast messages greater than 4 bytes in length.

For broadcast messages, 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.

5.1.1 Broadcast Messages and Power-Gating

Some systems may have inter-node dependencies on `Layer` power state, e.g. activating a higher power regulator before a high-power component. For this reason, by default nodes must not change `Layer` power state upon receipt of a broadcast message, excepting messages that explicitly change `Layer` power state.

Some nodes, for example a general purpose processor that is snooping, may want to wake their `Layer` for all messages. This is permitted, but nodes with non-standard broadcast power behavior must clearly document power semantics to be MBus compliant.

5.1.2 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.

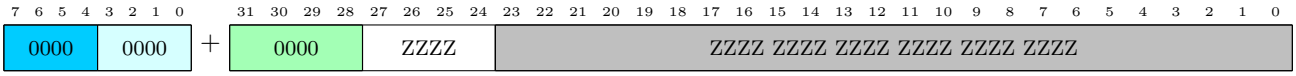
All MBus broadcast messages, except those sent on Broadcast Channels 2-7: Reserved, follow a common template. The messages are 32 bits long. The four most significant bits identify the message type/command. Some messages do not require all 32 bits. The unused bits are named *insignificant bits*. Messages may be truncated, omitting the insignificant bits on the wire[8].

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. The `Bus Controller` is responsible for handling Channel 0 messages. Channel 0 messages should not affect `Layer` power state.

[8] With the caveat that all MBus messages must be byte-aligned. Some insignificant bits may still be sent on the wire as a consequence.

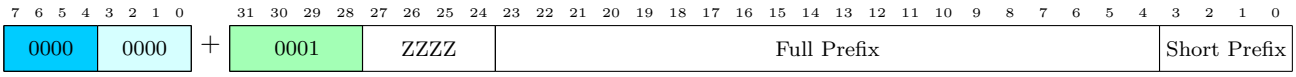
Query Devices



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



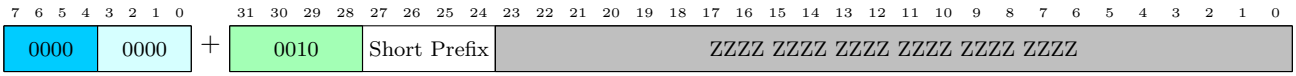
This message is sent in response to a Query Devices or Invalidate Prefix request. When responding to Query Devices, every node will be transmitting their address, and nodes should anticipate losing arbitration several times before they are able to send their response.

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 interjected.[9]

All nodes are required to support this message.

Enumerate Node

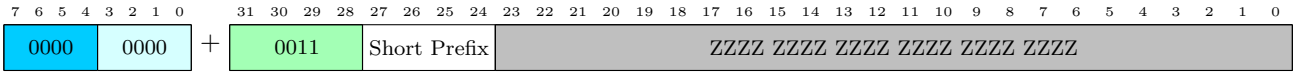


This message assigns a short prefix to a device. All nodes that receive this message and do not have an assigned short prefix **must** attempt to reply with a Query/Enumerate Response. Nodes with a short prefix shall ignore the message (broadcast NAK, that is, forward). Nodes shall perform exactly one 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.

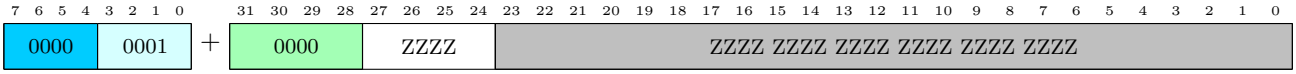
All nodes are required to support this message.

[9] An interjection should not occur during this message. Such an interjection would be an error.

Broadcast Channel 1: `Layer` Power Channel 1 is used to query and command the `Layer` 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 their lowest power state in response to an All Sleep message.

A node `Layer` is implicitly waked when a message is addressed to it, explicitly issuing a wake command is unnecessary to communicate with a node. Nodes may build finer-grained power constructs beyond the macro `Layer` control provided by MBus. For the purposes of MBus, a node's "sleep" state should be a minimal power state. Nodes may have different sleep configurations, e.g. different interrupts that are armed.

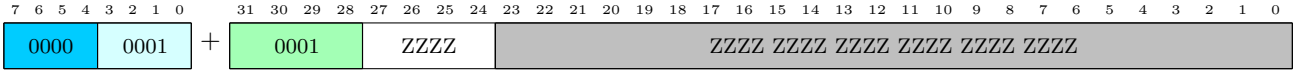
All Sleep



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*.

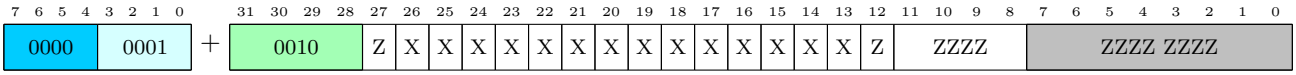
All power-aware nodes are required to support this message.

All Wake



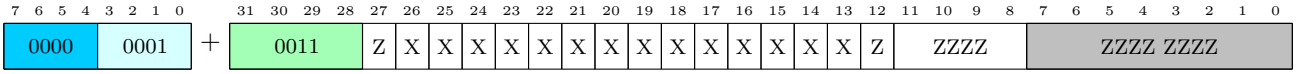
All nodes receiving this message **must** immediately wake up. The bottom 28 bits of this message are reserved and should be *ignored*.

Selective Sleep By Short Prefix



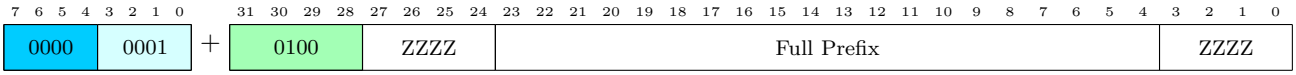
This message instructs selected nodes to sleep. The 16 bits of data are treated as a bit vector, mapping short prefixes to bit indicies. That is, the node with short prefix `0b1101` is controlled by the second bit received (bit 25 in the bit vector above). If a bit is set to `1`, the selected node **must** 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



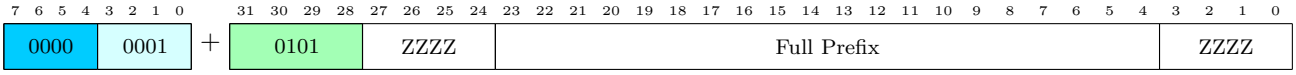
This message instructs selected nodes to wake. The 16 bits of data are treated as a bit vector, mapping short prefixes to bit indicies. That is, the node with short prefix `0b1101` is controlled by the second bit received (bit 18 in the bit vector above). If a bit is set to `1`, the selected node **must** 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 Channels 2-7: Reserved

5.2 Extension Messages (Address `0xffffffff`)

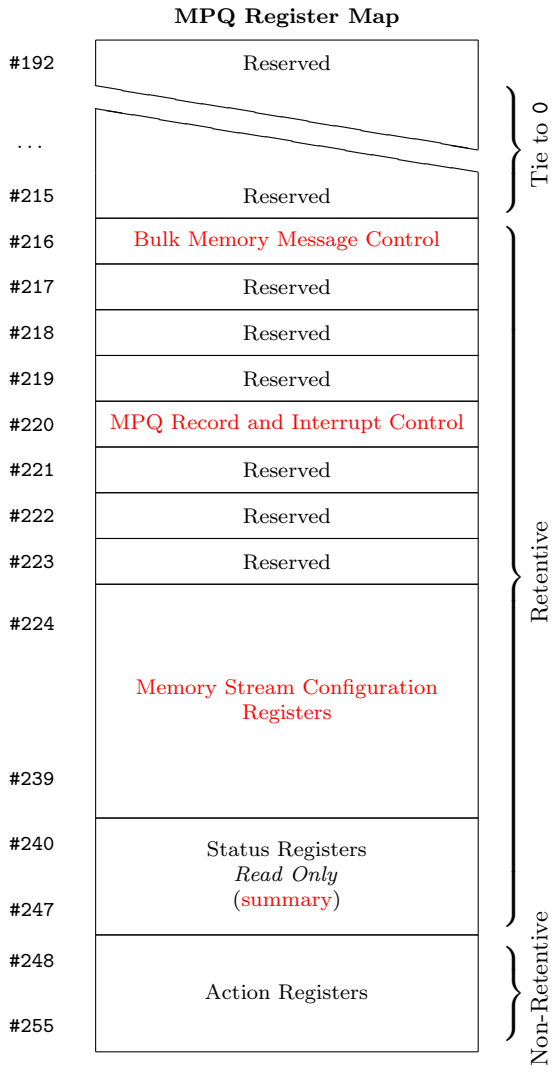
This address is reserved for future extensions to MBus.

6 MPQ: Point-to-Point Message Protocol

MBus also defines a common point-to-point messaging protocol: MPQ. Nodes are not required to support MPQ to be MBus compliant, however it is strongly encouraged.

MPQ defines two classes of data: register data and memory data. MPQ registers have 8 bit addresses and are 24 bits wide. MPQ memory has 32 bit addresses and stores data that is 32 bits wide. The register address space and memory address space are separate constructs, that is register address `0x00` need not map to memory address `0x00000000`, although aliasing is permitted.

6.1 MPQ Registers (Reg #192–255)



MPQ reserves the top 64 registers for control and configuration.

This space configures various MPQ capabilities. Much of the space is currently reserved for future expansion.

Attempts to write configuration for unsupported features (e.g. configuration of a memory streaming channel on a device with no memory) is undefined. Attempts to read unsupported features **must** NAK or return all `0`.

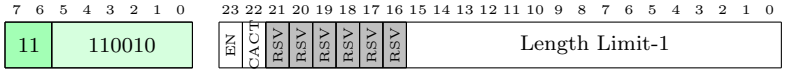
This section documents each of the registers. The top two bits of each MPQ register address are `11` followed by six bits that identify the type. All reserved bits should be treated as RZWI.

The MPQ controller stores state about its current operation in the status registers, #240–247. Each MPQ operation describes what the MPQ controller will **Record** and when it will trigger an interrupt. Reg #220: MPQ Record and Interrupt Control controls what events generate interrupts.

6.1.1 Reserved Registers

All registers not specified here are reserved. Writes to reserved registers should be ignored. Reads from reserved registers should return all `0`.

6.1.2 Reg #216: Bulk Memory Message Control



Def: `0x800000`, `0x000000` if no mem

The register controls the response of this chip upon the receipt of a Memory Bulk Write command.

- {`R[23]`}: `Enable (EN).`
  - Controls whether bulk memory transactions written to this device are enabled. If `EN` is `0`, a node must not modify the contents of memory in response to a bulk memory transaction.
  - **Acknowledement/Interjection:** The behavior of a bulk write when `EN` is `0` is undefined. Receivers are encouraged to interject and indicate receiver error, however they may exhibit any behavior, including ACK'ing the transaction and silently ignoring it.
- {`R[22]`}: `Control Active (CACT).`
  - If this bit is high, this register's length field acts as a limit for the maximum permitted bulk message length. A bulk message is allowed to write until this message limit is reached. If more data comes, the message **must** be NAK'd. The receiver should interject with receiver error as soon as it knows the length limit has been exceeded.
- {`R[15:0]`}: `Length Limit.`
  - The maximum permitted length in words of a memory bulk write message to any address.

6.1.3 Reg #220: MPQ Record and Interrupt Control

Default: `0x0000XX`

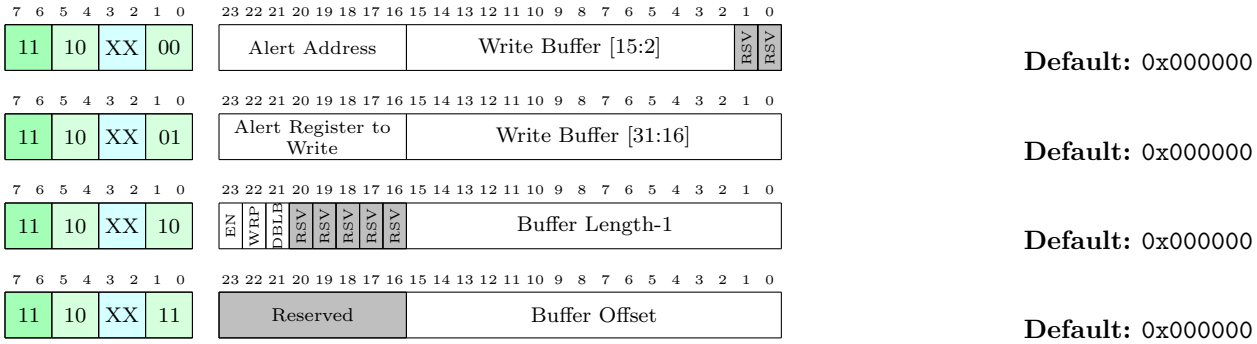
Each time a node completes a MPQ command, it checks the associated command `INT` bit. If high, the layer owner is interrupted.

Default Value: The default value is implementation dependent. It current defaults to all interrupts off (`0x000000`).

Note: See Finer-grained DMA control for possible future security considerations.

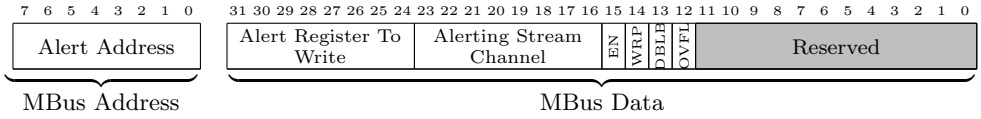
TODO: What are the ways in which a command that comes in the interrupt port from the layer should differ from a command that comes from the bus? Separate interrupt control?

6.1.4 Reg #224–239: Memory Stream Configuration



In MPQ each node has up to four independent, identical memory streaming channels. Each channel has two configuration registers. The two registers work together to configure each channel.

- {R01[15:0], R00[15:2], 2'b00}: Write Buffer.
  - Pointer to the beginning of a buffer in memory.
- {R00[23:16]}: Alert Address.
  - Defines where alerts are sent. If the alert prefix (bits 23:20) are set to the full-prefix indicator 1111, the alert is suppressed.
  - Alerts are sent whenever the end of the buffer is reached. If DBLB is active, an alert is also sent when the halfway point of the buffer is reached.
  - When a memory stream alert occurs, a node sends the following message:

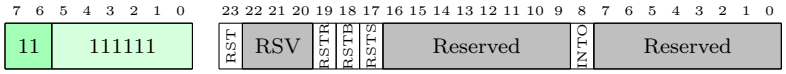


    - The MBus Address is set to the Alert Address specified by bits 23:16 in R1110XX00.
    - The top 8 bits of data are set to the Alert Register to Write specified by bits 23:16 in R1110XX01.
    - The next 8 bits are the Alerting Stream Channel, made up of this node's short prefix, followed by 01, followed by the channel that generated the alert—this should be the same address used to write to this stream channel.
    - The EN bit reports the current state of the EN bit of the alerting channel when the alert was sent.
    - The WRP bit is set if the write that generated this alert reached the end of the stream buffer.
    - The DBLB bit is set if the write that generated this alert reached the halfway point of the stream buffer and double-buffering is active for this stream.
    - The OVFL bit is set if the write that generated this alert reached the end of the stream buffer and there was already a pending alert with the WRP bit set or if the write that generated this alert reached the halfway point of the stream buffer and double buffering is active for this stream and there was already a pending alert with the DBLB bit set.
  - TODO: What happens if a node wishes to alert itself? e.g. a CPU that puts itself to sleep but wants to be interrupted when 1,000 samples have been written to memory?
- {R10[15:0]}: Buffer Length-1.
  - Defines the length of the buffer.
- {R10[23]}: Enable (EN).
  - Controls whether this channel is enabled. If EN is 0, a node must not modify memory in response to a memory stream message.[10]
  - **Acknowledement/Interjection:** The behavior of a stream write when EN is 0 is undefined. Receivers are encouraged to interject and indicate receiver error, however they may exhibit any behavior, including ACK'ing the transaction and silently ignoring it.

[10] Implementation Tip: Any undefined MPQ registers are defined to be RZWI, that is a read from an undefined register will read as all 0's (and a write ignored). Upon a read, this will return 0 for EN as required. Nodes that do not implement a memory stream channel do not require any logic to handle any memory stream messages, simply leave the register undefined.

- {R10[22]}: Wrap (WRP).
  - Defines node behavior when the end of the buffer is reached. If WRP is high, the Write Address Counter should reset to its original value. If WRP is low, the Write Address Counter value should be unchanged (it should thus be one past the end of the valid buffer) and EN should be set to 0.
- {R10[21]}: Double Buffer (DBLB).
  - Controls double-buffering mode. If double-buffering is active, the node should generate an alert halfway through the buffer in addition to at the end of the buffer. This mode is most useful when combined with WRP.
- TODO: Document Buffer Offset

6.1.5 Reg #255: Action Register



<No Storage>

This register requests that an action be performed. It is an error to request more than one action in a single request. Actions are processed from MSB to LSB, that is, if more than one action is requested *only* the highest priority action is actually taken.

A read from this register shall always return all 0.

A write of all 0 to this register shall perform no actions.

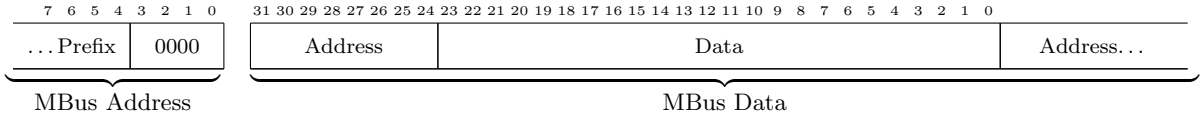
If any bit written to this register is non-zero, writing this register **must** be the last operation in the transaction. The behavoir of anything after a register action request in the same transaction is undefined.

- {R[23]}: Reset (RST).
  - Reset the entire node. The exact result of a reset is implementation-defined, however this request should be the most aggresive form of reset available.
- {R[19]}: Reset MPQ Registers (RSTR).
  - Reset MPQ configuration registers that affect register protocol behavior to their default value.
  - Resets #223–247.
- {R[18]}: Reset Bulk Registers (RSTB).
  - Reset MPQ configuration registers that affect memory bulk transfers to their default value.
  - Resets #242.
- {R[17]}: Reset Stream Registers (RSTS).
  - Reset MPQ configuration registers that affect memory stream transfers to their default value.
  - Resets #224–239.
- {R[8]}: Interrupt Owner (INTO).
  - Asserts the Layer Owner Interrupt.

6.2 Register Commands

MPQ register space is an 8 bit addressable array of 24 bit wide registers. Any undefined bits are treated as RZWI (Read as Zero, Write Ignored).

6.2.1 Register Write



Bits 0-23 of the MBus data field are written to the register addressed by bits 24-31. The write occurs immediately, as soon as the layer controller receives the message. Multiple registers may be written in a single MBus transaction by sending multiple data packets. Each 32 bit chunk of data is treated as if it were an independent transaction.

Record and generate interrupt after each register write.

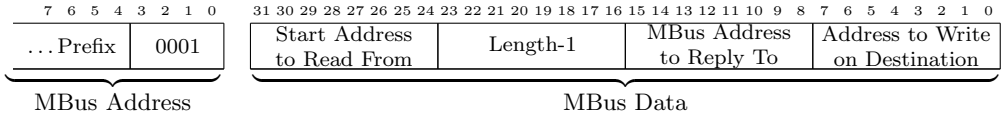
Reg #240:

Overflow: If more data is received after writing the last register (`0xff`), the destination address wraps and registers continue to be written, beginning at address `0x00`. *Tests:* **??**.

Unaligned Access: The behavior of a message ending on a non-word boundary is undefined.

Interjection Semantics: Each command is applied immediately when it is received. A four-command message would have a $4 \times 32 = 128$ bit data payload. If 63 bits are received prior to interjection, only the first command was applied. If 64 bits are received, the first two commands are applied. *Tests: A.3.1, A.3.2.*

6.2.2 Register Read



Bits 24-31 specify the address of the register to be read. Bits 16-23 may be used to request that multiple registers are sent. This field is a count of the number of values to be sent less one, that is a value of 0 requests 1 register is read and a value of 255 requests that all 256 registers are sent. Bits 8-15 are the MBus address the reply is sent to. Bits 0-7 specify the first address field of the Register Write response.

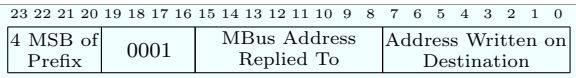
The response message is sent to the MBus address specified in bits 8-15 of the request and its data field is formatted exactly as the Register Write command: 8 bit address + 24 bit data. For reads of more than one register, the address field in the response is incremented by 1 for each register.

The response always sends the requested length. If a request for register #256 would have been made, the request wraps and begins from register #0. The destination register address wraps similarly.

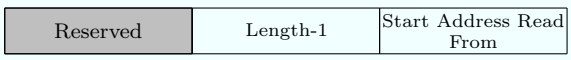
Note: The MBus address to reply to **must** be copied exactly. The FU_ID is not required to be Register Write. For example, if the FU_ID is Memory Stream Write, the effect is dumping the current register state to memory on the target addresss. *Tests: A.1.1.*

Record command information. Generate the whole response message. Interrupt after sending complete response.

Reg #240:



Reg #241:

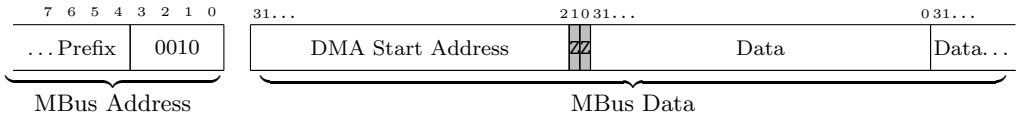


Interjection Semantics: If the reply is interjected, the transaction is aborted and is **not** retried.

6.3 Memory Commands

MPQ memory space is a 32 bit addressable array of 32 bit words of memory. Any undefined accesses are treated as RZWI (Read as Zero, Write Ignored). MPQ provides two types of memory commands: bulk and stream. A bulk memory transaction is a wholly self-contained event designed for DMA of large blocks of memory. Memory streams pre-configure the stream information in MPQ registers on the destination node and omit it from subsequent transactions, relying on the receiver to maintain and increment a destination pointer. Streams are useful for applications such as continuous sampling, where multiple, short messages are generated.

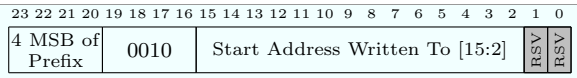
6.3.1 Memory Bulk Write



The first word received is the address in memory to begin writing to. The bottom two bits of the address field are reserved and **must** be transmitted as 0. Subsequent words are treated as data. The first word of data is written to the specified address. The next word of data is written to address+4 (the next word in memory) and so on. There is no limit on the length of this message.

Record command information. When the write completes, generate an interrupt. If an error occurs part way through, the Reg #250 should indicate the number of words actually written. TODO: Use reserved bit(s) to indicate error?

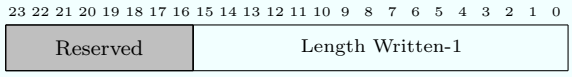
Reg #240:



Reg #241:

Reg #242, #243 not written.

Reg #244:



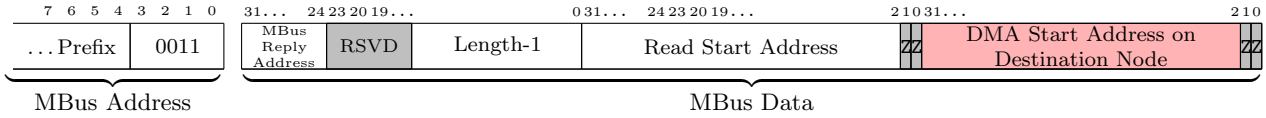
Implementation Note: These registers may be updated while the command is running, so long as they have the correct value once it completes. They are a good place to store the pointer and counter while the command is active instead of instantiating dedicated counter and address registers.

Unaligned Access: The behavior of a message ending on a non-word boundary is undefined.

Overflow: If more data is received after writing the last address in memory (`0xfffffffc`), the destination address wraps and data continues to be written, beginning at address `0x00000000`. *Tests: A.2.1.*

Interjection Semantics: Each word of data is written to memory immediately when it is received. A four-word message would have a $(1+4)\times 32 = 160$ bit data payload. If 95 bits are received prior to interjection, only the first word was written to memory. If 96 bits are received, the first two words were written to memory. *Tests: A.3.3, A.3.4.*

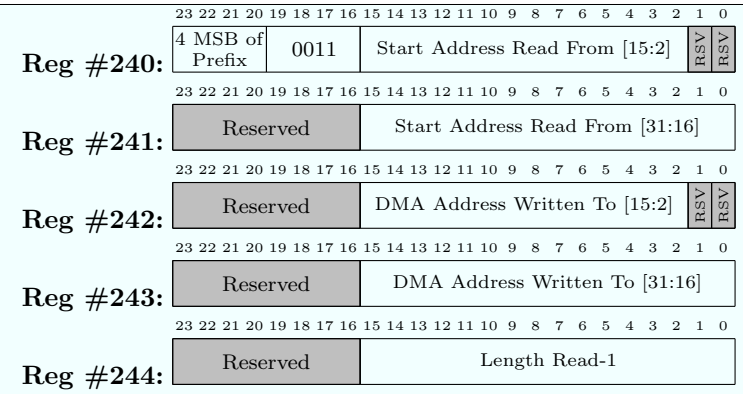
6.3.2 Memory Read



The first word indicates the MBus address to reply to and the length of the requested read in words less one. A length of 0 will reply with 1 word of data. The second word is the address in memory to read from. The bottom two bits of the address field are reserved and **must** be transmitted as 0.

The third word is **optional**. If the third word is present, it is prepended to the reply (generating a message with a Memory Bulk Write formatted payload). If the third word is omitted, data is immediately placed on the bus (generating a message with a Memory Stream Write formatted payload).

Record command information. Generate the response and send it. When the response completes, trigger an interrupt. If an error occurs part way through, the Reg #250 should indicate the number of words actually written. If the optional third word is omitted, Reg #251–252 are undefined. TODO: Use reserved bit(s) to indicate error?

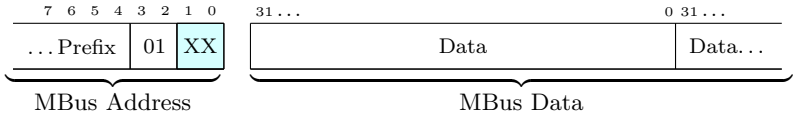


TODO: Message length isn't passed here, how to know if target address is included?

Overflow: If the starting address field and subsequent length exceed the memory space, that is a request for address `0x100000000` would have been made during the response, the layer controller wraps and continues sending from address `0x00000000`. *Tests: A.2.1.*

Interjection Semantics: If the reply is interjected, the transaction is aborted and is **not** retried.

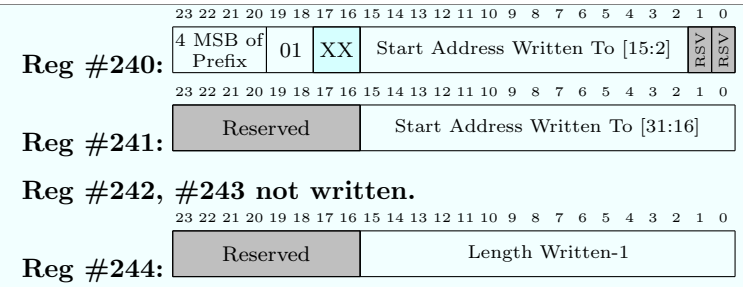
6.3.3 Memory Stream Write



MPQ nodes have up to four streaming memory channels. Each channel is controlled by configuration registers (Reg #224–239: Memory Stream Configuration). The destination of a memory stream write is specified by a combination of channel selection—the last two bits of the FU_ID—and the pre-arranged configuration.

The message payload is all data. The destination address is automatically incremented every time a word is written. There is no limit on the length of this message.

Record command information. When the write completes, generate an interrupt. If an error occurs part way through, the Reg #250 should indicate the number of words actually written. TODO: Use reserved bit(s) to indicate error?



Unaligned Access: The behavior of a message ending on a non-word boundary is undefined.

Overflow: If more data is received after writing the last address in memory (`0xffffffc`), the destination address wraps and data continues to be written, beginning at address `0x00000000`. *Tests:* **??**.

Interjection Semantics: Each word of data is written to memory immediately when it is received. A four-word message would have a $(1+4)\times 32 = 160$ bit data payload. If 95 bits are received prior to interjection, only the first word was written to memory. If 96 bits are received, the first two words were written to memory. *Tests:* **??**, **??**.

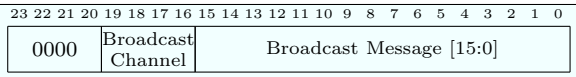
6.4 Broadcast Snooping

Broadcast Channel 0: Node Discovery and Enumeration and Broadcast Channel 1: `Layer` Power are built into the MBus protocol which runs below MPQ. However, it is possible that a MPQ node may wish to snoop broadcast traffic (in particular, Query/Enumerate Response).

If broadcast snooping is active TODO: Configuration register for this. Possibly some other filters such as channels?, whenever a broadcast message is received:

Record command information. Interrupt after message completes.

Reg #240:



Reg #241:

6.5 Undefined Commands

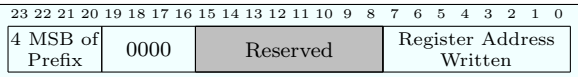
If command with an unknown FU_ID is received, MPQ behavior is completely undefined.

M3 Implementation Note: Current behavior silently drops messages with unknown FU_IDs.

6.6 Summary of Status Registers

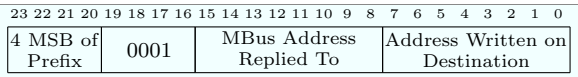
Record and generate interrupt after each register write.

Reg #240:

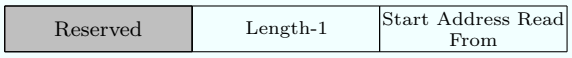


Record command information. Generate the whole response message. Interrupt after sending complete response.

Reg #240:

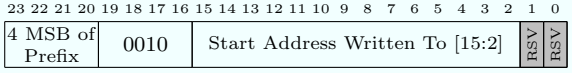


Reg #241:



Record command information. When the write completes, generate an interrupt. If an error occurs part way through, the Reg #250 should indicate the number of words actually written. TODO: Use reserved bit(s) to indicate error?

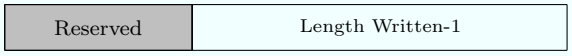
Reg #240:



Reg #241:

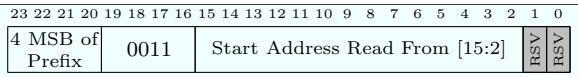
Reg #242, #243 not written.

Reg #244:



Record command information. Generate the response and send it. When the response completes, trigger an interrupt. If an error occurs part way through, the Reg #250 should indicate the number of words actually written. If the optional third word is omitted, Reg #251–252 are undefined. TODO: Use reserved bit(s) to indicate error?

Reg #240:



Reg #241:

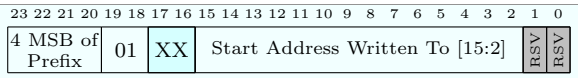
Reg #242:

Reg #243:

Reg #244:

Record command information. When the write completes, generate an interrupt. If an error occurs part way through, the Reg #250 should indicate the number of words actually written. TODO: Use reserved bit(s) to indicate error?

Reg #240:



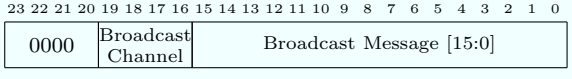
Reg #241:

Reg #242, #243 not written.

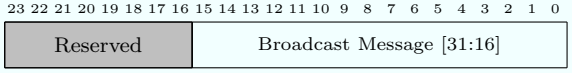
Reg #244:

Record command information. Interrupt after message completes.

Reg #240:



Reg #241:



7 MPQ Programmer's Model

This section documents the MBus and MPQ MMIO interface used in the M3 system.

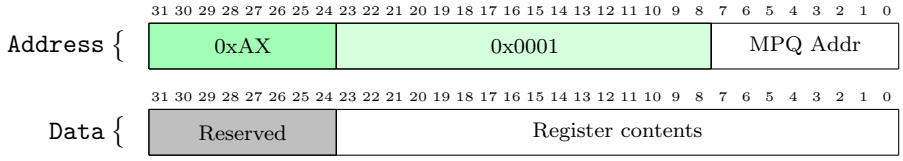
7.1 Accessing MPQ Registers [simple method]

The MPQ registers are mapped directly into the MMIO address space. Since MPQ registers are only 24 bits wide, the top 8 bits of data are always RZWI.

7.1.1 MPQ Registers

These registers map directly to MPQ registers. For details of each register's behavior, see MPQ Registers (Reg #192–255). Registers below #192 are chip-specific registers.

Bits Required	Purpose
Address	
24	Memory Map Location
8	MPQ Register
Data	
24	Register contents



7.2 Accessing MPQ Registers [efficient method]

This is just a thought. It reduces a lot of masking and shifting on the CPU side, but requires the HW to access either 2 or 4 registers on each MMIO operation.

As MPQ registers are 24 bits wide and machine words are 32 bits wide, we can map 4 MPQ registers onto 3 MMIO words. We define a *bank* of MPQ registers as a set of four MPQ registers. Each bank has a `TOP`, `LOW`, and `HIGH` register.

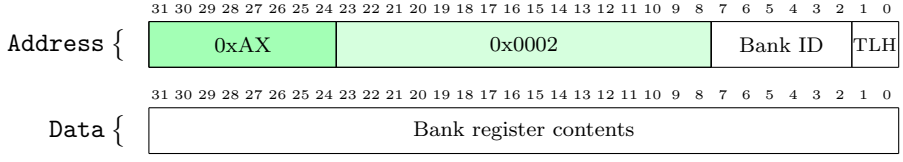
```
BANK0_T: {R3[23:16], R2[23:16], R1[23:16], R0[23:16]}
BANK0_L: {R1[15:0], R0[15:0]}
BANK0_H: {R3[15:0], R2[15:0]}
```

There are 256 MPQ registers, and thus 64 banks.

7.2.1 MPQ Registers

These registers map directly to MPQ registers. For details of each register's behavior, see MPQ Registers (Reg #192–255). Registers below #192 are chip-specific registers.

Bits Required	Purpose
Address	
24	Memory Map Location
6	Bank ID
2	Top/Low/High
Data	
32	Bank register contents



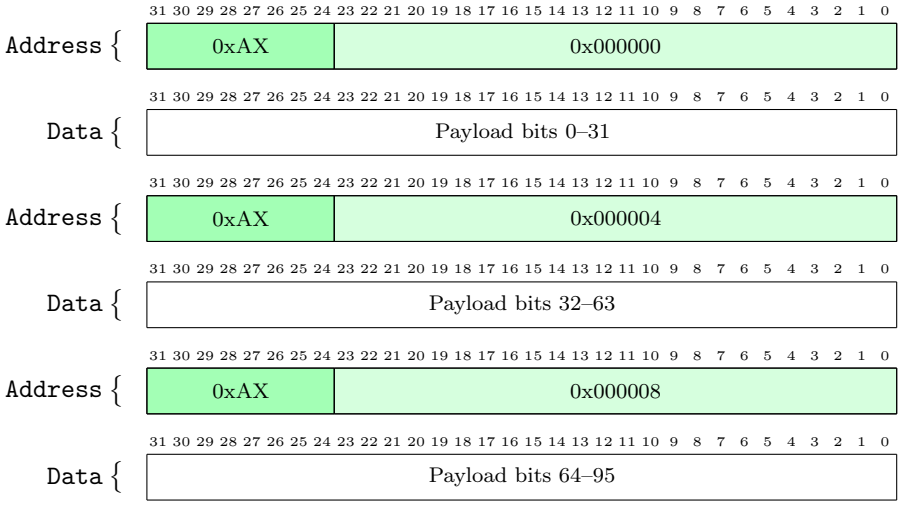
7.3 Sending MBus Transactions

The core issues MBus transactions by emulating incoming MBus transactions. For example, to issue a DMA transaction to another node (a memory *write*), the core emulates a memory *read* request from the destination node. This can be tricky to reason about, especially for writing general-purpose transactions. System designers are highly encouraged to use the provided `mbus.c` library.

7.3.1 `CMD0, CMD1, CMD2`

These three registers hold the data "from" the bus. Bits "arrive in order", that is, the MSB of `CMD0` is the first *data* bit to arrive on the bus. The address is not included. The destination prefix is omitted, and the FUID is specified by the `FUID_LEN` [write only] register.

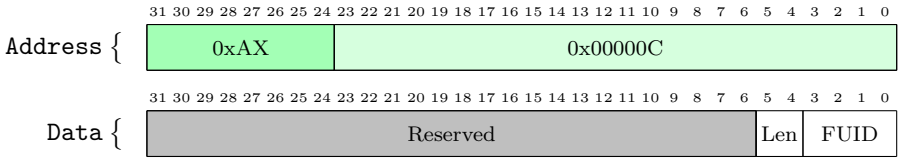
Bits Required	Purpose
Address	
32	Memory Map Location
Data	
32	Payload



7.3.2 `FUID_LEN` [write only]

This register specifies the FUID of the received message and the length of the `MBUS_CMD` that is valid. A write to this register will issue the transaction.

Bits Required	Purpose
Address	
32	Memory Map Location
Data	
4	FUID
2	Length (in words)



7.4 Interrupts

Reg #220: MPQ Record and Interrupt Control controls what events generate interrupts. Interrupt handlers should read Reg #240[23:16] to determine the interrupt source.

8 Specifications

8.1 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 **??**).

8.2 Endianness: Byte-Level

MBus sends from Byte 0...Byte N.

8.3 Endianness: Bit-Level

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

8.4 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 8.12 Interjection Rules for details on the minimum uninterruptable message length.

8.5 Minimum Maximum Message Length

All MBus mediators 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 mediator node's `CLK_IN` port up to and including the last bit latched (Request Interjection in Figure 5). The counter shall be an *inclusive* counter, that is the mediator node shall not interject until it receives the $N + 1$th bit.

Commands to query and configure the upper bound are specified in Broadcast Channels 2-7: Reserved.

8.6 Retransmission

There is no required 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 implementations and/or software.

8.7 Minimum Buffer Size

All MBus nodes are required to support both short and full addresses (8.10 Addressing). All MBus nodes are further required to accept a minimum of 32 bits (4 bytes) of data in an individual transmission.

8.8 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** interject the current transmission and indicate a receiver error (Control Bits: `01`).

The receiver must interject 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_INTERJECT 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 interjection, but no later than that such that it is assured that it will interject the transmission before the transmitter attempts to interject to signal end of message even if the receiver is of lower priority. XXX This should be a specified test

8.9 Clock Speed

TODO: What is appropriate specification for clock speeds? I2C has defined 'speed-classes'; SPI is a free-for-all; UART & co has semi-agreed on bauds...

8.10 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 4.3 Short Prefix Assignment and 4.2 Full Prefix Assignment respectively. Addresses `0x00`, `0x01`, and `0xffffffff` are reserved.

8.11 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.

8.12 Interjection Rules

MBus permits any node to interject 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 interjection. An exception is made for the transmitting node, which is permitted to send messages shorter than 32 bits.

Nodes wishing to interject 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 interjecting node must wait until it has latched the 33rd bit of data before attempting to interject.[11]

8.13 Failed Arbitration (Spurious Wakeup)

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

The mediator node **must not** immediately interject the bus, however. As laid out in **?? ??**, the arbitration edges may be used as input to node sleep controllers. To avoid potential issues related to half-waking power gated nodes, the mediator node must wait until Begin Transmission before interjecting the bus.

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

[11] If the node instead attempted to interject on the 32nd bit and was a topologically higher-priority node, it would not permit the transmitter to signal End of Message.

9 ToDo

9.1 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.

9.1.1 Automatically fragment long MPQ messages

Add a configuration register for a maximum message length, maybe one for each message type, that limits maximum transaction length. Requests to send messages over this length should be automatically fragmented by the sender.

9.1.2 Full Address Support in MPQ

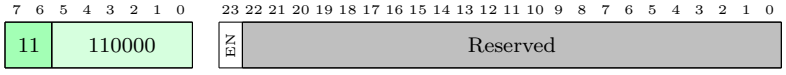
Currently there is no way to generate a response that sends a message to a full address. A full prefix is 24 bits long. The following are some possible methods to support full addresses:

1. **Use new FU_IDs.** Easiest solution, but uses a lot of the remaining FU_IDs. Could do something creative such as the full prefix (24 bits) is sent and then the next 3 or 4 bits map the remaining words onto one of the existing FU_IDs. This wastes at least the 4 bits that previously specified the short prefix.
2. **Full Prefix Register(s).** Store a full prefix in a register. Since registers are also 24 bits, they can hold a full prefix exactly. This is actually a little unfortunate, since a destination FU_ID also needs to be specified. While the short prefix `1111` in the current message can be used as a sentinel, something needs to indicate which register the full prefix should be read from. The FU_ID field could be repurposed to this end, but then there is nothing to specify the FU_ID of the eventual message. This could spill over to two registers, but that is an inefficient use of registers. Full prefix registers also become another contested, shared resource with this scheme.
3. **Optional Extra Word.** Use the existing short prefix as a sentinel. If set to `1111`, the (next, last) word will be the full prefix. Since the short prefix is the first 4 bits, could also just "replace" the first word of the message with a full address—this has nice symmetry with how short/full prefixes are interpreted on the bus. In any of these cases, an extra 8 bits become available for some other purpose. The runtime variable length message can be tricky to implement. Injecting an extra word into the middle of a message makes it harder for simple devices with fixed send buffers to support these messages.

9.1.3 Finer-grained DMA control

Currently DMA is all-or-nothing. This is how pretty much all M0 DMA controllers work, but it's not the best. It would be neat to explore an IOMPU concept (analagous to A-series IOMMUs). Another option could be to pass the request to the core as an interrupt, but I actually think that's more complicated.

9.1.4 Reg #240: Register Message Control



Default: `0x800000`

- {`R[23]`}: `Enable (EN).`

  Controls whether register writes to this device are enabled. If `EN` is `0`, attempts to write registers #0–247 are silently ignored.

  Note: Registers #248–255 (`b'11111XXX`) can always be written.

  Acknowledement/Interjection: Since a single register write transaction could write both enabled and disabled registers, a node should not interject when a disabled register is written. A node should ACK to indicate that the transmission was received successfully, even if nothing is actually written. *This behavior is subject to change in future revisions of MPQ.*

  Warning: If `EN` is disabled, it cannot be re-enabled by a write to this register since register writes are disabled. Access can be recovered using Reg #255: Action Register, however this will reset all registers.

TODO What about a local write (e.g. a register write via an interrupt from the CPU)? Should probably always allow those. Does that make sense as an exception?

10 Document Revision History

- Revision 1.0-alpha – June 14, 2015
  Initial Public Release

A Test Cases

This section documents some of the trickier corner cases in MBus design. This is by no means a comprehensive list of tests, but each of these cases should be tested to ensure compatibility across MBus implementations.

For most tests, we recommend that nodes are assigned a Static Short Prefix Assignment so that enumeration can be skipped.

A.1 Registers

A.1.1 Dump Register Content to Memory and Restore to Other Registers

`A) 4 Nodes: N1 -> N2 -> N3 -> MED (-> N1)`

This test dumps a node's registers to the memory of another node by requesting a register read that issues a streaming memory write. Some of these dumped registers are then written to different registers on another node by using a streaming memory read that issues a register write.

1. Begin with N1's register data full of known, non-zero values. N2 and N3 registers and memory should be zeroed.
2. N2 configures the memory stream address for N3:
   ??: 0x3X + 0xXXXXXXXX
3. N2 requests that N1 copy 19 registers to N3's memory. Choose register addresses such that both read and destination addresses wrap.
   ??: 0x11 + 0xED_12_3X_0xF8
4. Verify the memory contents of N3.
5. N1 requests that 9 of the registers are written to N2:
   ??: 0x??

A.2 Memory

A.2.1 Bulk Transaction Overflow Wrapping

`A) 3 Nodes: N1 -> N2 -> MED (-> N1)`

This test verifies both basic memory bulk transfer operation and that the bulk transfers handle overflow wrapping correctly.

1. Begin with N1's memory full of known, non-zero values. N2 memory should be zeroed.
2. N1 sends a request to N2 to read the last word and first 3 words of its memory and writes it to the last 2 words and first 2 words of N1's memory.
   ??: 0x25 + 0xfffffffc_12_000011_ffffffff8
3. Verify the correct values from N2 are written to memory in N1.

Test Name:	Success:
mem-bulk-overflow-A	- Write 4 words to N2

A.3 Interjection Timing

A.3.1 Third-Party Interjector for Many Registers

`A) 4 Nodes: RX -> INJ -> TX -> MED (-> RX)`

Have the TX node send a register write command that writes at least 3 registers long. The INJ node should generate an interjection on the 63rd bit of **data** with control bit 0 set to `!EoM`. The effect should be that the transmitter sees 63 data clock edges and the receiver sees 65 data clock edges. At the end of the transaction, the first register should be written, the second register should not. The TX layer controller should believe that the transmission failed and it has written either 7 bytes or 1 word (depending on implementation), but not 8 bytes or 2 words.

Repeat this test, interjecting on the 64th edge. The result should be the same.

```
B) 4 Nodes: TX -> INJ -> RX -> MED (-> TX)
C) 4 Nodes: TX -> RX -> INJ -> MED (-> TX)
```

Repeat the same test with these node positions.
Repeat these tests, interjecting on the 64th edge. The result should be the same.

```
D) 4 Nodes: INJ -> RX -> TX -> MED (-> INJ)
```

Repeat the same test but interject at the 64th, 65th, and 66th data edges. For 64 and 65, the result should be the same. For 66, two registers should be written.

```
Test Name:          Success:
inj-reg-long-A-63 - Write 1 register
inj-reg-long-A-64 - Write 1 register
inj-reg-long-B-63 - Write 1 register
inj-reg-long-B-64 - Write 1 register
inj-reg-long-C-63 - Write 1 register
inj-reg-long-C-64 - Write 1 register
inj-reg-long-D-63 - Write 1 register
inj-reg-long-D-64 - Write 1 register
inj-reg-long-D-65 - Write 1 register
inj-reg-long-D-66 - Write 2 registers
```

A.3.2 Third-Party Interjector at Last Register

This test is the same as the previous, except the TX node should attempt to write exactly 2 registers.

```
Test Name:         Success:
inj-reg-end-A-63 - Write 1 register
inj-reg-end-A-64 - Write 1 register
inj-reg-end-B-63 - Write 1 register
inj-reg-end-B-64 - Write 2 registers
inj-reg-end-C-63 - Write 1 register
inj-reg-end-C-64 - Write 2 registers
inj-reg-end-D-63 - Write 1 register
inj-reg-end-D-64 - Write 1 register
inj-reg-end-D-65 - Write 2 registers
inj-reg-end-D-66 - Write 2 registers
```

A.3.3 Third-Party Interjector for Long Memory Bulk Transfer

Repeat the same tests as Third-Party Interjector for Many Registers, adding an address to the MBus data payload and interjecting 32 cycles later such that either one or two words are written to memory.

```
Test Name:          Success:
inj-mem-bulk-long-A-63 - Write 1 word
inj-mem-bulk-long-A-64 - Write 1 word
inj-mem-bulk-long-B-63 - Write 1 word
inj-mem-bulk-long-B-64 - Write 1 word
inj-mem-bulk-long-C-63 - Write 1 word
inj-mem-bulk-long-C-64 - Write 1 word
inj-mem-bulk-long-D-63 - Write 1 word
inj-mem-bulk-long-D-64 - Write 1 word
inj-mem-bulk-long-D-65 - Write 1 word
inj-mem-bulk-long-D-66 - Write 2 words
```


A.3.4 Third-Party Interjector at End of Memory Bulk Transfer

Repeat the same tests as Third-Party Interjector at Last Register, adding an address to the MBus data payload and interjecting 32 cycles later such that either one or two words are written to memory.

```
Test Name:         Success:
inj-mem-bulk-end-A-63 - Write 1 word
inj-mem-bulk-end-A-64 - Write 1 word
inj-mem-bulk-end-B-63 - Write 1 word
inj-mem-bulk-end-B-64 - Write 2 words
inj-mem-bulk-end-C-63 - Write 1 word
inj-mem-bulk-end-C-64 - Write 2 words
inj-mem-bulk-end-D-63 - Write 1 word
inj-mem-bulk-end-D-64 - Write 1 word
inj-mem-bulk-end-D-65 - Write 2 words
inj-mem-bulk-end-D-66 - Write 2 words
```