MBus and M3 Processor Layer Family (Version 20) (PRCv20, PRCv20G, PREv20, PREv20E)

Yejoong Kim < yejoong@umich.edu,yejoong@cubeworks.io>[1, 2]

[1]Michigan Integrated Circuits Laboratory, University of Michigan, Ann Arbor
[2]CubeWorks, Inc., Ann Arbor, Michigan

Revision 1.0

Last Updated: May 24, 2019

Contents

Part I

MBus

1 Before Diving In...

Most of the description about MBus in this chapter is taken from *MBus Specification, Revision 1.0-alpha*, written by Pat Pannuto et al.

Note that MBus implementation has gone through several revisions to fit the current M^3 design environment and requirements. As a result, there are some discrepancies, which includes module names, signal names, and some behaviors, between this document and *MBus Specification*. This document reflects all the updates and changes made so far at the time of this writing. Some of the original figures have been re-drawn reflecting the updated names and behaviors.

You can access the original *MBus Specification* document at `http://mbus.io`.

The MBus implementation described in this document is MBus Release 5.

2 Introduction

2.1 MBus Overview

MBus is an ultra-low power system bus. The original design was motivated by the Michigan Micro Mote (M^3)[1]. The goal of MBus, however, is to be a general purpose bus for hyper-constrained systems. MBus requires four pins per node, uses only static CMOS 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, thus suitable for low-power systems. MBus requires one more capable node to act as a bus mediator node, whose primary duties are providing the MBus clock and mediating arbitration. This mediator node is also called a master node.

Note that, in this document, every details of MBus will be explained in the context of M^3 systems.

2.2 M^3 System Overview

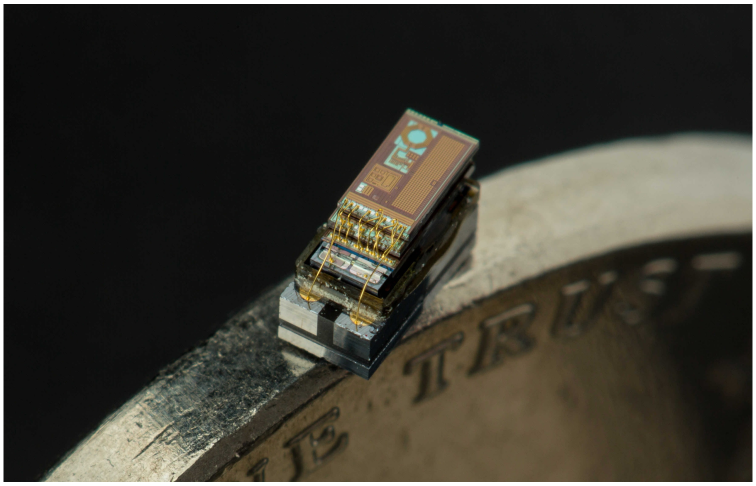


Figure 1: M^3 Pressure Sensing Stack System

M^3 (Michigan Micro Mote) is a modular cubic-millimeter-scale wireless sensor node platform developed by Michigan Integrated Circuits Laboratory (MICL). M^3 can be regarded as a successor of prototypes presented in the following publications:

> Mingoo Seok, et al., "The Phoenix Processor: A 30pW Platform for Sensor Applications," *Symp. VLSI Circuits*, 2008.
>
> Gregory Chen, et al., "A Millimeter-Scale Nearly-Perpetual Sensor System with Stacked Battery and Solar Cells," *ISSCC*, 2010.
>
> Gregory Chen, et al., "A 1 Cubic Millimeter Energy-Autonomous Wireless Intraocular Pressure Monitor," *ISSCC*, 2011.
>
> Yoonmyung Lee, et al., "A Modular 1mm^3 Die-Stacked Sensing Platform with Optical Communication and Multi-Modal Energy Harvesting," *ISSCC*, 2012.

There are many other publications on which each particular M^3 component is based on. See **??** for the list of such publications.

[1] See Section 2.2.

One of the typical M^3 systems is shown in Figure 1, which measures around 1mm × 3mm × 2mm. This example system consists of 3 layers of integrated circuit chips: near-field radio chip, power-management/processor chip, and a sensor chip. In addition to this, there is a 2uAh battery and a MEMS pressure sensor, completing the stack system.

In most cases, M^3 system is called a *stack*, because it is a stack of several chips and components. Thus, in this document, these two words, *system* and *stack*, are regarded as synonyms and are used interchangeably.

Also, each chip in the whole stack is called a *layer*, because it is a layer of the stack. It can be also referred as a *node*, because it behaves as a node in the MBus context. Thus, in this document, these three words, *layer*, *chip*, and *node*, are regarded as synonyms and are used interchangeably.

Main characteristics of M^3 systems are summarized below.

2.2.1 Modular Design and MBus

Each layer can be designed in different technology. For example, an analog sensor layer can be designed in relatively old 180nm process to benefit from low mismatch and high voltage margin. On the other hand, a digital circuit layer can be designed in advanced technologies to utilize low dynamic power and higher performance. A flash memory layer could be designed in a process technology that provides an embedded flash option. Each layer is connected through MBus. This modular approach also means that a layer can be easily replaced with another layer as needed during the design time. For example, assume a stack system has been already built with 1Mb flash layer. If there is a need for larger data storage, then building another stack system with a larger flash layer is very easy by replacing the 1Mb flash layer with the one with more flash capacity. This great flexibility allows M^3 systems to be used in various applications.

It is especially beneficial for student designs. When a student designs, for example, a new analog circuit, they may choose to add the MBus interface in the chip. Then it can be easily integrated into an M^3 system, which already has a processor, power-management unit, radio, etc. This allows more realistic testing/usage scenarios and helps write stronger papers.

2.2.2 Low-Power and Duty-Cycled Operation

In most cases, M^3 systems are heavily duty-cycled, meaning that they stay in an extremely low-power mode (called Sleep mode) for most of their lifetime, and waking up only for a short time (called Active mode) to handle high power operations such as analog measurements or wireless transmission.

During Sleep mode, the system can harvest energy from ambient sources like visible or infrared light. Since the system stays in Sleep mode for most of its lifetime, the Sleep mode power (sleep power) is a significant (if not dominant) part of the system power consumption. Usually, a layer may consume less than 1nW during its Sleep mode, which sums up to a few nW of sleep power in a complete system consisting of several layers.

During Active mode, the power constraint becomes relatively loose, and it is limited by the driving capability of power-management unit (PMU) layer or battery internal resistance. In most cases, 10–50uA current draw is not a problem, and a higher current draw, such as 200uA, is also possible through proper configuration of PMU layer.

In some applications, the system consumes more power during Sleep mode. This includes the case where the system keeps running motion detection or more accurate timer during Sleep mode. Usually, these applications require a bigger battery to sustain such operations.

2.2.3 Small Batteries

M^3 systems include one or more batteries. Since the overall system volume is very limited, the battery size must be also tiny. The smallest battery used in M^3 system was measured only 1mm × 2.2mm × 0.2mm and its capacity was only 2uAh. Other moderate-sized systems use batteries ranging from 4uAh to 16uAh, but all of them are in mm-scale. Hence, the internal resistance of the batteries are relatively high, ranging from 5kOhms to 30kOhms, and it could be a substantial limit on peak current draw, especially when designing a high-power[2]analog circuits. A bigger battery can be used for such high-power systems. An example would be a Seiko coin-cell battery which measures 7.9mm (diameter) × 2.6mm (height), with 34mAh capacity with 100Ohms internal resistance. However, it is still very small and low capacity, compared to other commercially available batteries. All circuits designed for M^3 system must take this into account.

The batteries used in M^3 system has an open-circuit voltage (V_{OC}) ranging from 1V to 4.4V. The PMU layer has a selection switch to determine the voltage level required for system boot-up. By default, it is set to 4V, meaning that the battery voltage must be higher than 4V to be able to boot up the system. It can be configured to 1V, which enables the use of batteries with $<$4V V_{OC}.

2.2.4 Layer Name

Each layer must have its own and unique name. Conventionally, the layer name consists for three alphanumeric letters followed by a version indicator 'v', and a version number. Optionally, there can be a sub-version number/letter following the version number. This layer name is called `LNAME` in short.

`LNAME` consists of all upper-case letters, except the version indicator 'v'. An lower-case version of `LNAME` is called `lname`.

Usually, the first three alphanumeric characters in `LNAME` represents the type of the layer. Thus, it is often called `LTYPE` in short.

Examples of the layer name are shown below in Table 1 to help better understanding.

Table 1: M^3 Layer Name Example

Layer Name	`LTYPE`	`LNAME`	`lname`	Description
PRCv20	PRC	PRCv20	prcv20	Processor layer version 20
FLPv3L	FLP	FLPv3L	flpv3l	Flash layer version 3 (large)
N2Nv1	N2N	N2Nv1	n2nv1	Node-to-Node radio version 1

2.2.5 Master and Member Layer

Each layer in a M^3 system is connected through MBus. MBus requires two signals to be transmitted, clock and data. It is a daisy-chain-like configuration, hence each layer has four digital pads dedicated for MBus: namely, clock-input (`CIN`), clock-output (`COUT`), data-input (`DIN`), and data-output (`DOUT`). Figure 2 shows a top-view of the wirebonding of MBus wires in a M^3 stack consisting of 4 layers, where the arrow indicates the signal flow.

In MBus specification, there are two types of layers: master[3]and member. A master layer is the one which drives the MBus clock signal and arbitrates the priority if any conflict arises among the member layers. The other layers in the stack are member layers. An M^3 stack must have only one master layer.

Master layer has a clock generator in it, which generates the MBus clock. This clock generator runs whenever the system is in Active mode, and it clocks the MBus FSM[4]inside the master layer in Active mode. It transmits the clock signal through its `COUT` pad only when necessary, which means, other member layers see the clock signal coming into their `CIN` pad only when necessary. In idle state, the clock wires are held high. If there is a case where more than one member layers request to send MBus messages, the master layer must handle the arbitration. This functionality is implemented in the MBus FSM inside the master layer.

[2] This indicates a high power in the context of M^3 systems. Generally, anything larger than ∼ 10uAh is regarded as *high power* in this context.

[3] Master layers are also called Mediator layers

[4] This includes MBus Bus Ctrl and MBus Master Ctrl. See Figure 3.

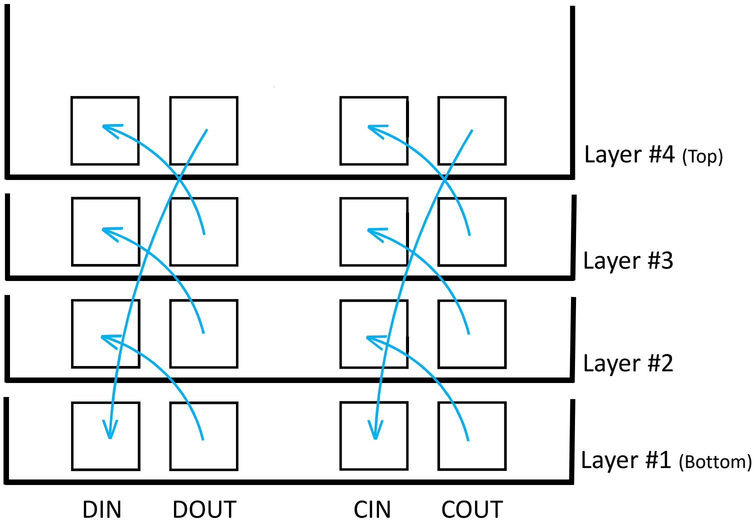


Figure 2: MBus Wires in M^3 System This shows a top-view of the wirebonding of MBus wires in an example M^3 stack, consisting of 4 layers.

Member layer does not have a clock generator for the MBus clock. Hence, its MBus FSM[5]is clocked by the incoming MBus clock signal through its `CIN` pad. Usually, the member layer forwards the incoming clock signal (`CIN`) to the clock output (`COUT`). However, there is a certain scenario where the member layer stops forwarding the clock, which will be described in a later section. Note that, the member layer should have a clock generator for its other purposes. Usually it has a clock generator for Layer Ctrl (See Figure 4).

In most cases, MBus data wires are driven by a layer who is sending the MBus message. This layer is called *TX Layer*. Usually the MBus message is targeted for a specific layer, and this layer is called *RX Layer*. The TX Layer must toggle its `DOUT` at the falling edge of its `CIN`, and the RX Layer must sample its `DIN` at the rising edge of its `CIN`. Other layers are just forwarding their incoming data (`DIN`) to the data-output pad (`DOUT`), and these layers are called *FWD Layers*. There are certain scenarios where the RX Layer drives the data wire. This will be discussed in a later section.

Since the inception of M^3 systems, processor layers have been the master layer. Processor layers include PRC (Processor layer) and PRE (Processor Extended layer), and their sub-versions. This means that every M^3 stack designed so far has one of PRC or PRE layer.

In this document, PRCv20 Family indicates a group of PRC and PRE layers, version 20.

Table 2 shows a list of commonly used M^3 layers. In this table, Type 'N/A' indicates that the layer does not have an MBus module inside. Usually they just have power/ground pads and do not have digital interface.

2.2.6 Supply Voltages and Ground

Typically, each layer needs three supply voltages, namely `VDD_3P6`, `VDD_1P2`, and `VDD_0P6`. As the names suggest, their nominal voltage levels are 3.6V, 1.2V, and 0.6V, respectively[6].

MBus FSMs require `VDD_1P2` supply. `VDD_0P6` is required to detect the power-on-reset sequence. Thus, these two supply voltages are mandatory.

If not used elsewhere, `VDD_3P6` is primarily used for ESD detection and Dirty VDD. If the layer does not have any circuit running at `VDD_3P6`, then the layer can use `VDD_1P2` for the ESD detection and Dirty VDD. It may degrade the ESD detection capability, but it saves one double-sized pad space and a wirebonding.

[5] This includes MBus Bus Ctrl and MBus Member Ctrl. See Figure 4.

[6] In reality, when used with Power-Management Unit layer, the actual voltages are around $\sim$ 4.0V, $\sim$ 1.35V, and $\sim$ 0.65V.

Table 2: List of Commonly Used M^3 Layers (as of May 2019)

Layer Name	Type	Process	Description
PRCv20	Master	TSMC 180nm	Processor Layer
PREv20	Master	TSMC 180nm	Processor Extended Layer
PMUv11	Member	TSMC 180nm	Power-Management Unit Layer
HRVv9	Member	TSMC 180nm	Harvester Layer
MEMv3	Member	TSMC 180nm	Additional Memory Layer
MRRv10A	Member	TSMC 180nm	Medium-Range Radio Layer
RDCv4	Member	TSMC 180nm	Resistance-to-Digital Converter Layer
SNTv4	Member	TSMC 180nm	Sensor and Timer Layer
SRRv7	Member	TSMC 180nm	Short-Range Radio Layer
SOLv7	N/A	TSMC 180nm	Solar Cell Layer
DCPv13S	N/A	TSMC 180nm	De-coupling Capacitor Layer
LDCv1A	N/A	TSMC 180nm	Light-to-Digital Converter Layer
BSRv2	N/A	TSMC 180nm	Back-Scattered Radio Layer
FLPv3L	Member	TSMC 90nm	Flash memory (8Mb)
FLPv3S	Member	TSMC 90nm	Flash memory (1Mb)
VIMv1	Member	TowerJazz 65nm	VGA Imager Layer
N2Nv1	Member	Fujitsu 55nm	Node-to-Node Radio Layer

Power-Management Unit layer (PMU) is a special layer that generates these three supply voltages and provide them to other layers. A battery is directly connected to the PMU layer, so this becomes additional supply voltage to PMU, which is named `VDD_BAT`[7]. If non-PMU layers need direct battery access, then they could also have `VDD_BAT` pad to get the voltage directly from the battery.

Every layer needs a shared ground, called `VSS`.

[7] As mentioned previously in Section 2.2.3, the battery voltage may range from 1V to 4.4V.

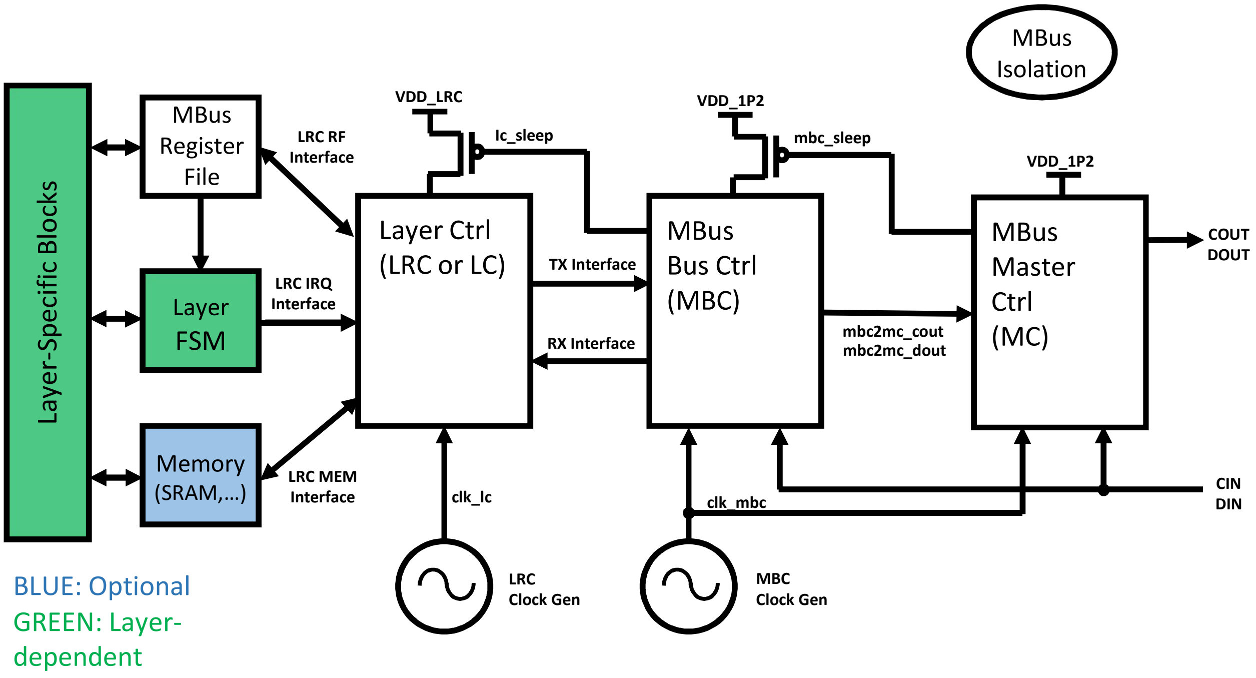


Figure 3: Master layer block diagram

3 MBus Layer Design

MBus defines two *physical* types of layers: a master layer and member layers. An instantiation of MBus must have one and only one master layer and must have at least one member layers (N≥1). The maximum number of member layers is a function of clock speed[8]as well as the available address space. In most cases, M^3 systems utilizes MBus Short Prefix[9], and the maximum number of member layers is 13. Including the master layer, there can be up to 14 layers in an M^3 system, in such case. If a system also utilizes MBus Full Prefix[9], it can have 2^{20} layers in theory.

During a transmission, MBus defines three *logical* types of layers: a transmitting layer (TX Layer), a receiving layer (RX Layer), and forwarding layers (FWD Layers). During a transmission, there must be exactly one TX Layer and one RX Layer, unless it is a broadcasting message, where multiple RX Layers are listening. Any number (N≥0) of FWD Layers are permitted.

3.1 Physical Design

The physical design of a member and master layer is the same, each must expose a `DIN`, `DOUT`, `CIN`, and `COUT` pad.

3.1.1 Master Layer

A master layer requires 4 signals: `DIN` (Data In), `DOUT` (Data Out), `CIN` (Clock In), and `COUT` (Clock Out).

[8] Obviously, the minimum MBus cycle time is (2 × Number of layers × layers-to-layers latency). MBus designers are obligated to consider all possible 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.

[9] Short Prefix and Full Prefix are explained in detail in Section 6. However, the current MBus implementation does not support the use of Full Prefix in `MBUS_ADDR`.

Figure 3 shows a typical chip block diagram of a master layer. Note that green-colored blocks are layer-specific; it is not a part of MBus specifications and there is no restriction put by MBus in how it can be implemented, except the LRC IRQ Interface (explained in the next paragraphs).

Layer Ctrl Layer Ctrl (*LRC* or *LC* in an abbreviated form) is a part of MBus implementation, and it serves a role as an interface between layer-specific blocks (green color in the figure) and the other MBus blocks. Layer Ctrl is in a power domain where its power is controlled by `lc_sleep` signal. In most cases, its power supply (`VDD_LRC`) is `VDD_1P2`. However, it is also design-dependent, and some layers use a lower voltage (e.g., `VDD_0P6`) to achieve power reduction.

Usually, a Layer FSM (as shown in the figure) should be designed so that it communicates with Layer Ctrl using the LRC IRQ Interface (Section 8.1). This interface is used whenever the Layer FSM wants to generate and send an MBus message to other layers. Note that, it is an uni-directional interface (Layer FSM → Layer Ctrl). If Layer Ctrl needs to control Layer FSM, it needs to done indirectly, using LRC RF Interface or LRC MEM Interface[10].

MBus Register File MBus Register File is an essential block in any MBus implementation. Only Layer Ctrl can write into MBus Register File using LRC RF Interface. Usually this happens when this layer receives an MBus Register Write message. If, for example, Layer FSM needs to write into MBus Register File, it must be done through LRC IRQ Interface, where Layer FSM provides necessary information to Layer Ctrl using LRC IRQ Interface, and then Layer Ctrl writes into MBus Register File using the provided information. LRC IRQ Interface is explained in detail in Section 8.1. In most cases, MBus Register File is in an always-on `VDD_1P2` power domain, but it is also possible to have power-gated (non-retentive) registers or registers in multiple power domains. See Section 9 for more details.

Memory A layer can have a memory (e.g., SRAM), and Layer Ctrl supports it through LRC MEM Interface. It has 32-bit-wide address space, which is word-aligned, where 1 word represents 32 bits. Thus, theoretically it supports up to 4GB, but it is also limited by the power/space budgets. The memory can be directly interfaced with Layer FSM, which requires some kind of arbitration block (not shown in the figure).

LRC Clock Gen Note that Layer Ctrl requires its own clock generator (LRC Clock Gen in the figure). Its clock, `clk_lrc`, can be also used to clock Layer FSM and the memory (if any), which essentially makes LRC IRQ Interface and LRC MEM Interface synchronous.

MBus Bus Ctrl MBus Bus Ctrl (*MBC*) is often called MBus FSM[11], and its main role is to decode and create MBus messages. Whenever there is an incoming MBus message, MBus Bus Ctrl performs serial-to-parallel conversion; it samples `DIN` at `CIN`'s rising edges, gathers 32-bit data, and then delivers it to Layer Ctrl through RX Interface. Whenever Layer Ctrl needs to send an MBus messages (probably triggered by Layer FSM through LRC IRQ Interface), Layer Ctrl delivers the necessary information to MBus Bus Ctrl through TX Interface, and then MBus Bus Ctrl generates corresponding clock (`mbc2mc_cout`) and data (`mbc2mc_dout`) signals, which then become `COUT` and `DOUT`.

MBC Clock Gen In a master layer, MBus Bus Ctrl is clocked by its own clock generator (MBC Clock Gen in the figure). Its clock, `clk_mbc`, clocks MBus Bus Ctrl in general, and MBus Bus Ctrl has its internal clock gating to generate (`mbc2mc_cout`) only when necessary. As a result, `COUT` is a clock-gated version of `clk_mbc` and has the same frequency and phase. Note that `clk_mbc` and `clk_lrc` do not need to be synchronous to each other. Therefore, TX Interface and RX Interface must implement asynchronous hand-shake. MBus Bus Ctrl is in a power domain where its power is controlled by `mbc_sleep` signal, and its power supply is `VDD_1P2`.

[10] LRC RF Interface and LRC MEM Interface are explained in detail in a separate document, *MBus Implementation Guide* (pending).

[11] Strictly speaking, MBus Master Ctrl and MBus Member Ctrl are also a part of MBus FSM.

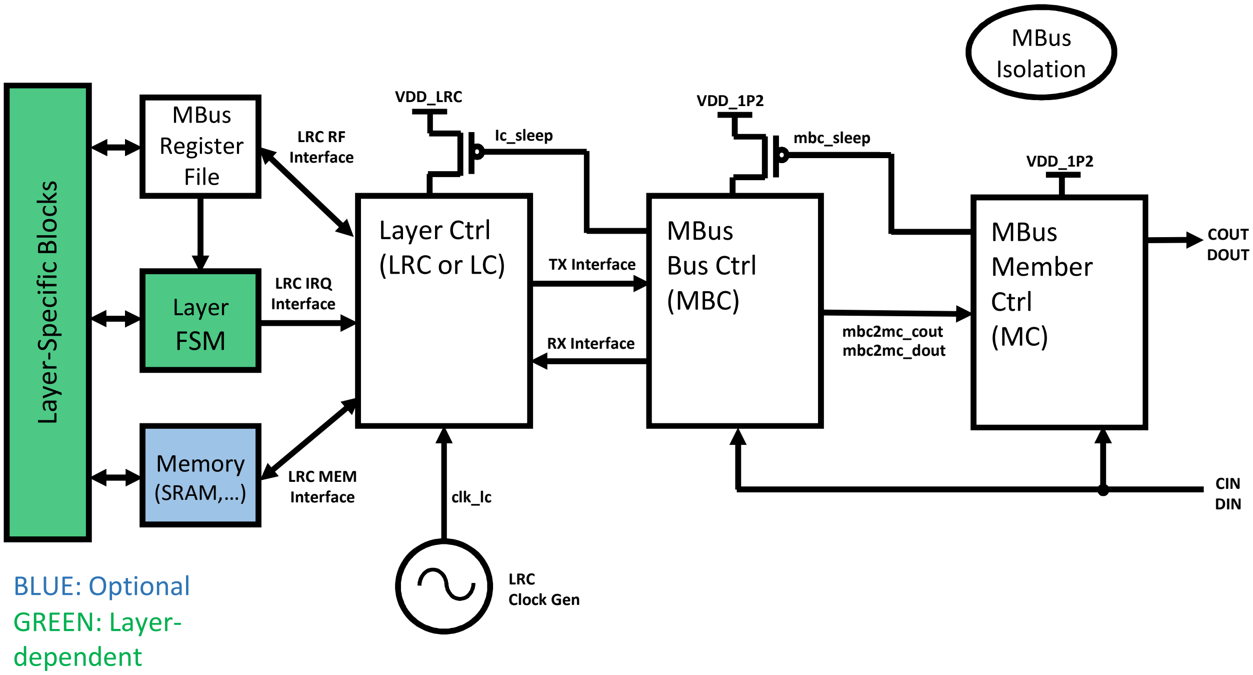


Figure 4: Member layer block diagram

MBus Master Ctrl MBus Master Ctrl is a part of MBus FSM, and its main role is to implement behavior shown in Figure 6. In addition, it also generates signals for system power control and MBus exceptions. It must be in always-on `VDD_1P2` power domain. It directly drives the `COUT` and `DOUT` pad. Hence, in M^3 systems, MBus clock and data are at `VDD_1P2` level.

MBus Isolation Finally, there is an MBus isolation module. Various MBus blocks (e.g., Layer Ctrl, MBus Bus Ctrl, etc) are in their own power domains, which are not necessarily same, so there must be a proper isolation scheme implemented. Usually, MBus isolation module is in always-on `VDD_1P2` power domain. Sometimes, MBus isolation module also includes level conversions, if there are blocks running at different voltages. For example, if Layer Ctrl is in `VDD_0P6` power domain, while MBus Bus Ctrl is in `VDD_1P2` power domain, then TX Interface should include proper level conversion and isolation.

3.1.2 Member Layers

A member layer requires 4 signals: `DIN` (Data In), `DOUT` (Data Out), `CIN` (Clock In), and `COUT` (Clock Out).

Figure 4 shows a typical chip block diagram of a member layer. Note that green-colored blocks are layer-specific; it is not a part of MBus specifications and there is no restriction put by MBus in how it can be implemented, except the LRC IRQ Interface (as explained previously).

Most of the member layer block diagram is similar to the master layer counterpart. One of the few differences is that, member layers do not have the MBC Clock Gen. Thus, in member layers, the role of `clk_mbc` in master layer is handled by `CIN`. As noted earlier, `CIN` is a clock-gated version of `clk_mbc`, which means that member layers have `CIN` toggling only when there is an MBus activities (i.e., when there is an MBus message transaction). This results in a low power consumption during `IDLE` state as there is no clock activity when there is no MBus activity.

This also indicates that an M^3 system cannot consist of member layers only. Also, there must be only one master layer in an M^3 system; otherwise, the multiple MBC Clock Gens from multiple master layers would cause conflicts.

IDLE state is a part of FORWARDING state (See Figure 7). Most of the time, a member layer is in the FORWARDING state. While forwarding, a member layer must amplify and forward signals from the DIN pin to the DOUT pin and from the CIN pin to the COUT pin. Designs should attempt to minimize latency between these pins, as this latency becomes the upper limit of the maximum MBus speed.

3.1.3 Bus Connections

- DIN, DOUT
  - DATA pins are connected in a round-robin fashion; the DOUT of one layer is connected to the DIN of the next.
  - The connection of DATA lines forms a loop as shown in Figure 5.
  - There are no requirements for the placement of layers in the DATA loop, but the ordering will have an impact on bus arbitration. See Section 4.2 for more details.
- CIN, COUT
  - CLK pins are connected in a round-robin fashion; the COUT of one layer is connected to the CIN of the next.
  - The connection of CLK lines forms a loop as shown in Figure 5.
  - The connection of CLK lines matches that of DATA lines. That is, if a layer N$_a$ connects its DOUT to the DIN of layer N$_b$, the COUT of N$_a$ is also connected to the DIN of N$_b$.

An MBus system must have a minimum of two layers (one master layer and one member layer). Single layers that are not connected to a bus should tie CIN and DIN high and leave COUT and DOUT floating.

In most cases, layers just forward their CIN to COUT, hence each CIN and COUT pair just looks same[12]. Thus this document uses a term 'CLK' or 'CLK line' when there is no need to distinguish CIN and COUT. In this case, 'CLK' or 'CLK line' indicates each wire connection between COUT and CIN across layers.

Similarly, when needed, 'DATA' or 'DATA line' is used to indicate the wire connections between DOUT and DIN across layers.

3.2 Logical Design

There are three *logical* types of MBus layers: transmitting, receiving, and forwarding. MBus can be considered multi-master, as any layer is capable of transmitting to any other layer. The master layer adopts these same three personalities, but its behavior differs slightly during arbitration (Section 4.2).

[12] Of course, there is a slight delay from CIN to COUT in each layer, but this delay is—or *should be*— negligible in practice.

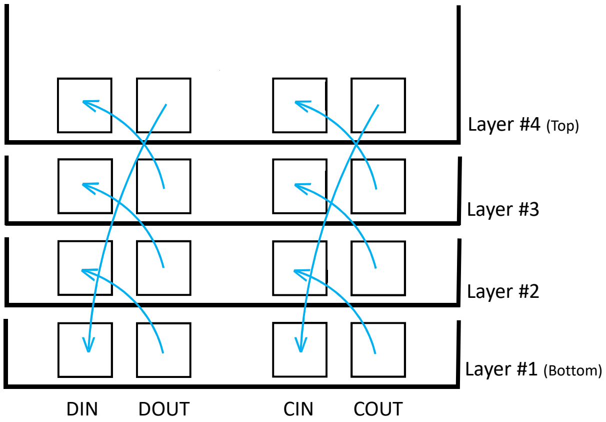


Figure 5: MBus Physical Topology This figure is same as shown in Figure 2, and it also serves as an example showing high-level picture of MBus physical design. One of the layers must be a master layer, while others are member layers. Member layers and a master layer are connected in a loop, with `DATA` and `CLK` lines forming independent rings.

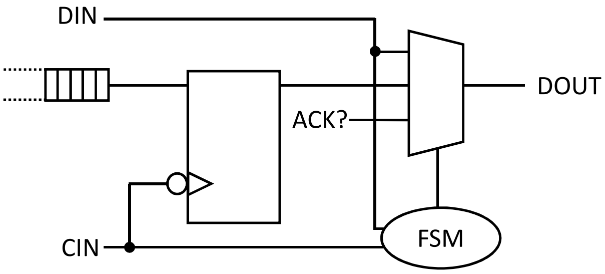


Figure 6: MBus Logical Model The Finite State Machine selects between the three modes a layer can be in. Top: *forwarding*, Middle: *transmitting*, Bottom: *acknowledging*. This model omits some of the subtleties of arbitration, for simplicity.

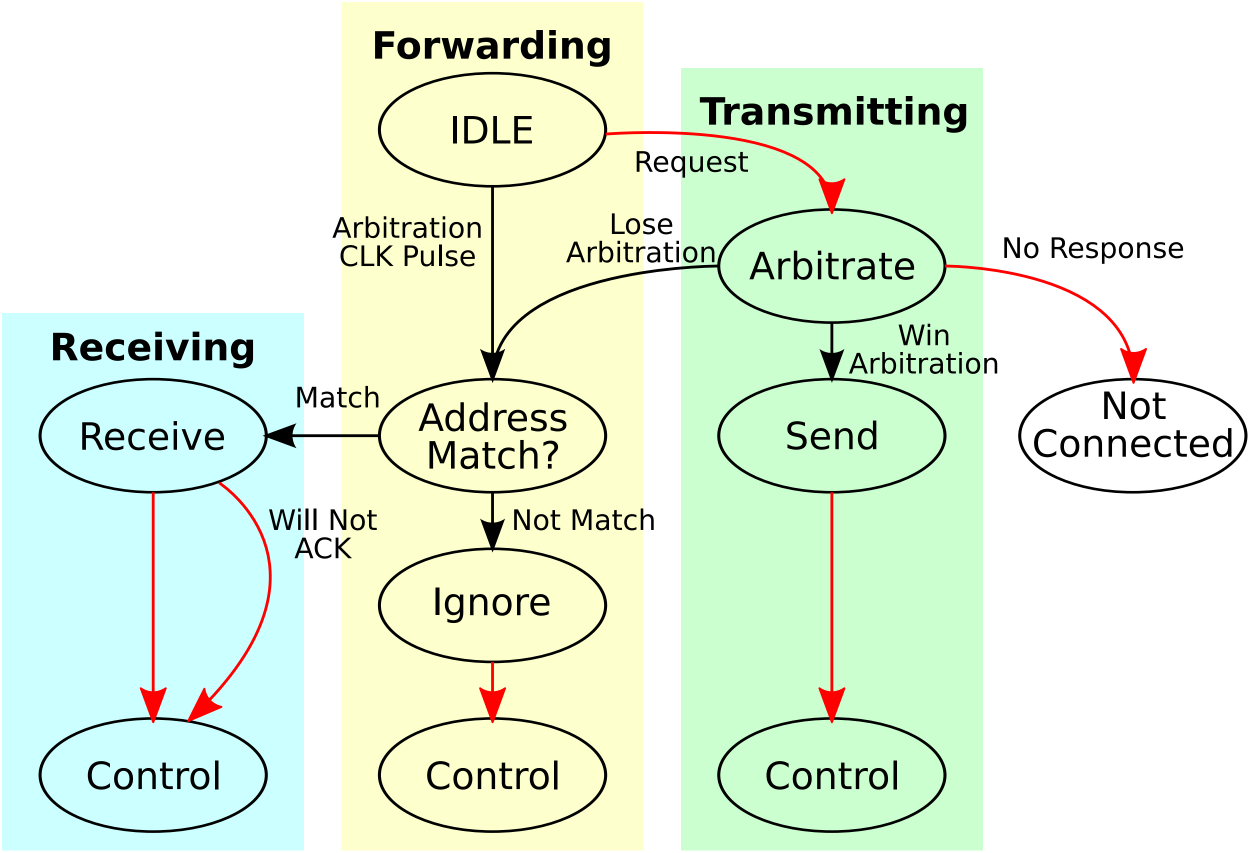


Figure 7: FSM describing the high-level logical behavior of MBus layers. Black arrows indicate transitions that occur on bus clock edges. Not shown are implicit arrows from any state to FORWARDING.CONTROL. From any CONTROL state, layers progress to IDLE state.

3.2.1 Forwarding

Forwarding is the most common state for all MBus layers. Observe in Figure 6 the very simple, short logic path from `DIN` to `DOUT`.

FORWARDING.IDLE This is the rest/idle state for MBus layers. In this state, member layers may be completely power-gated and asleep. The only obligation is that `DIN` is forwarded to `DOUT`, and `CIN` is forwarded to `COUT`.

- Master Layer Exceptions
  - In FORWARDING.IDLE state, the master layer does not forward `DIN` to `DOUT`.

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

FORWARDING.IGNORE In this state, layers simply forward data. Layers remain in this state until an interjection happens.

3.2.2 Transmitting

TRANSMITTING.ARBITRATE A layer initiates a transmission by pulling its `DOUT` line low. A layer may only attempt to initiate a transmission while the bus is idle.

The TRANSMITTING.ARBITRATE state is left at the next rising edge of the `CLK` line[13]. If a member layer's `DIN` is high on the rising `CLK` edge, it has won arbitration. A layer that loses arbitration should begin listening to see if it is the destination layer. If the `CLK` line never goes high — where *never* is defined as four times the MBus clock period — the layer should consider itself as disconnected.

Details of priority arbitration are omitted here for simplicity. See Section 4.2 for details.

- Master Layer Exceptions
  - The master layer always wins arbitration. Note that when this occurs, the master layer's `DIN` will be low.
  - If a master layer pulls its `DOUT` low and its `DIN` never goes low, it should be considered NOT_CONNECTED.

TRANSMITTING.SEND During TRANSMITTING.SEND, a TX Layer pushes bits onto the bus as described in Section 4.3. A TX Layer completes its transmission by interjecting (Section 4.4) the bus and indicating the transmission is complete.

TRANSMITTING.CONTROL As a TX Layer, a layer is responsible for the first control bit. This bit is set by a TX Layer to indicate that the complete message has been sent. The TX Layer must listen to the subsequent control bits to establish if the message was acknowledged and if the targeted layer (RX Layer) is sending a response.

[13] The master layer handles the `CLK` transitions.

3.2.3 Receiving

RECEIVING.RECEIVE A RX Layer is also obligated to forward data along the bus. In the sketch shown in Figure 6, during the receiving process, a receiving layer remains in pass-thru mode until an interjection happens.

RECEIVING.CONTROL A RX Layer must acknowledge the successful receipt of a transmission. If a RX Layer 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.

3.2.4 Exception States

NOT_CONNECTED A robust implementation should include detection of some kind for an attempt to utilize the bus when the layer is not actually connected to a bus (so that it may report failure)[14]. After a layer pulls its `DOUT` low, there is a maximum possible duration before a master layer must pull `CLK` low in response. If `CLK` is not pulled low, the layer should consider itself disconnected and report failure to send as appropriate.

[14] This exception handling is not implemented in the current MBus implementation.

4 MBus Bus Design

During normal operation, MBus remains in Bus Idle (Section 4.1). A transmission begins with Arbitration (Section 4.2), then Message Transmission (Section 4.3). The TX Layer then goes into Interjection (Section 4.4) and indicates the complete message was sent. During the Control Bits (Section 4.5) period, acknowledgment is negotiated. After Control Bits, MBus returns to Idle. RX Layer may optionally send a response message.

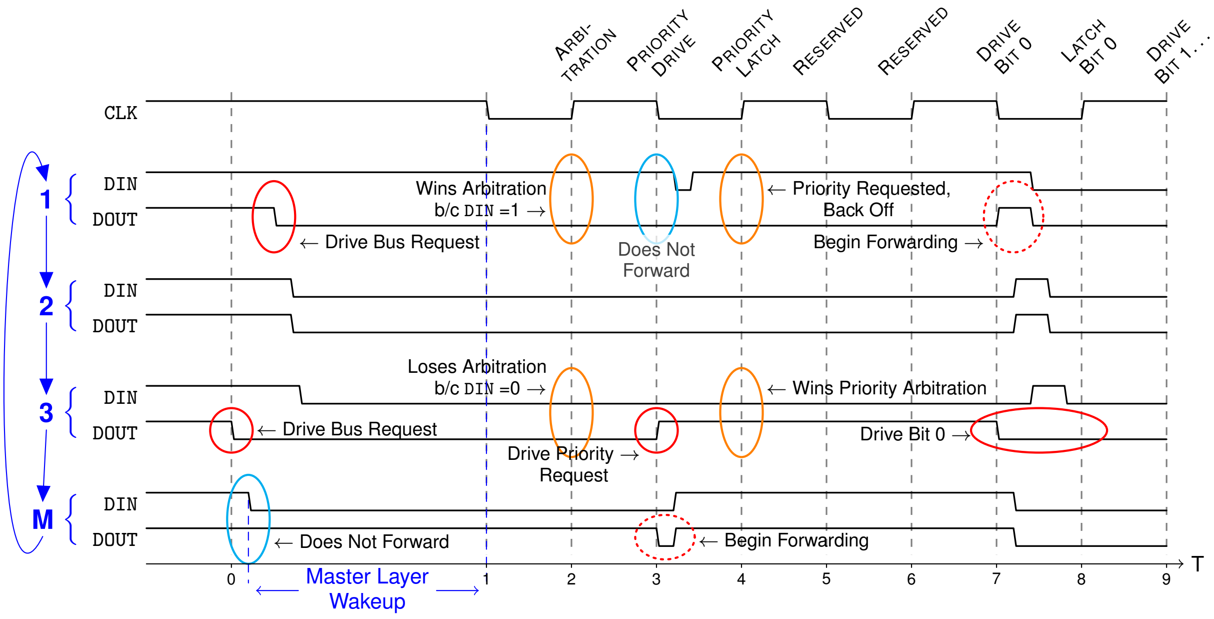


Figure 8: MBus Arbitration. Layer numbers in the left side (along with the blue arrows) indicate the order of MBus connection. Note that 'M' indicates the master layer. To begin a transaction, one or more layers pull down on DOUT. Here it shows layer 1 and layer 3 requesting the bus at nearly the same time (layer 1 shortly after layer 3). Layer 1 initially wins arbitration, but Layer 3 uses the priority arbitration cycle to claim the bus. The propagation delay of the DATA line between layers is exaggerated to show the shoot-through nature of MBus. Momentary glitches caused by layers transitioning from driving to forwarding are resolved before the next rising CLK edge.

4.1 Bus Idle

In Bus Idle, all lines (CLK and DATA) are high. All member layers are in forwarding state and the master layer is waiting to begin arbitration.

4.2 Arbitration

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

To request to transmit on the MBus, a layer should pull its DOUT line low. All member layers remain in forwarding state during arbitration, thus when their DIN line goes low, they are obligated to pull their DOUT line low. The master layer, however, does **NOT** pull its DOUT line low in response to its DIN line going low. When the master layer's DIN line goes low, the master layer first initiates the wakeup for the set amount of time ('Master Layer Wakeup' in Figure 8)[15]. Once the master layer fully wakes up (T=1), it will pull the CLK line low and hold it low for some period (T=1 $\sim$ 2). During IDLE state (and thus arbitration), member layers must forward the CLK signal.

By the end of the period, at T=2, the effect of the arbitration is achieved. A member layer is in one of three possible states:

1. CLK low, DIN high, DOUT high: Lost arbitration (didn't participate)
2. CLK low, DIN high, DOUT low : Won arbitration
3. CLK low, DIN low, DOUT low : Lost arbitration (either lost, or didn't participate)

The rising edge of CLK at T=2 commits the results of arbitration and all participating member layers advance their state machines appropriately.

If the master layer wishes to transmit on the bus, all member layers will be in the third state (i.e., CLK low, DIN low, DOUT low). Note this arbitration protocol introduces a *topology-dependent priority*. Firstly, the master layer has a greater priority than any member layer as its DOUT will propagate around the entire DATA loop. The priority of the member layers is inversely related to their proximity to the master layer in the DATA loop. That is, the furthest layer from the master layer (i.e., the layer whose DIN is connected to the master layer's DOUT) has the highest priority of member layers. The closest layer to the master layer, (i.e., the layer whose DOUT is connected to the master layer's DIN) has the lowest priority. In Figure 8, Layer M (master layer) has the highest priority, followed by Layer 1, Layer 2, and Layer 3.

At T=2, the master layer drives the CLK high. The normal arbitration phase ends on this rising edge. The next two CLK edges (T=3 and 4) allow for a high-priority arbitration. As MBus priority is topology-dependent, it provides a priority arbitration cycle to allow a low priority member layer to preempt transmissions. This priority mechanism is still topology-dependent, that is, if Layer 1 in Figure 8 had also attempted to send a priority message, it would have won that arbitration as well. The priority arbitration cycle is two CLK edges, at T=3 and 4; the falling edge (T=3) is for layers to drive their priority requests onto the MBus and the rising edge (T=4) is used to latch the results. During priority arbitration, the layer that won the original arbitration **must not** forward its DIN to DOUT. At the end of priority arbitration (T=4), if the original winner did not request a priority message, the layer must sample its DIN line. If the DIN line is still low, it means there was no priority message; if the DIN line has gone high, however, the layer has lost priority arbitration and must back off to allow the priority message requester to transmit.

Whichever layer that ultimately won arbitration transitions from a FWD Layer to a TX Layer. Layers that lost arbitration revert to FORWARDING.ADDRESS_MATCH.

4.3 Message Transmission

The falling edge after arbitration (T=7 in Figure 8) is defined as "Begin Transmission". On this edge, the TX Layer should switch its DOUT mux from forwarding DIN to the transmit FIFO. DOUT is expected to be valid and stable during every subsequent rising CLK edge.

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

- Matches: Layer promotes itself from FORWARDING.ADDRESS_MATCH to RECEIVING.RECEIVE.
- Does Not Match: Layer transitions from FORWARDING.ADDRESS_MATCH to FORWARDING.IGNORE.

There is no protocol-level delineation between address and data bits. The TX Layer sends address + data as a continuous stream of bits (for details on MBus addressing, see Section 6). Once a TX Layer has sent the complete transmission, it interjects the bus.

[15] This is required only when the master layer was in Sleep mode before it detects its DIN going low. However, in the current MBus implementation, the master layer always goes through this 'Master Layer Wakeup' delay regardless of the previous Sleep/Active state.

Legal transmissions on MBus must be byte-aligned. The requirement allows receiving layers, such as Layer 1 and Layer 3 in Figure 9, to disambiguate the significance of the last two bits received. A legal transmission will end cleanly on a byte boundary if the layer topologically follows the transmitter (e.g. Layer 3 in Figure 9) or will end on a byte+2-bit boundary if the layer topologically precedes the transmitter (e.g. Layer 1 in Figure 9).

4.4 Interjection

During normal operation, interjection is only permitted after the first 32 bits of data have been transmitted. The interjection mechanism, however, is also the MBus reset mechanism[16], and as such member layer interjection detectors **must** always be active whenever a layer is not Idle.

An MBus interjection is defined as a series of pulses on the DATA line while the CLK line remains high. After N pulses where $N \geq 3$, a layer enters interjection[17]. Edges on the DATA line while the CLK is low are ignored. This permits potentially racy changes of DOUT drivers to safely change on the falling edge of the CLK without potentially introducing spurious interjections.

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

4.4.1 Nested Interjections

In normal operation, interjections are not permitted to “nest”, that is, the MBus may not be interjected while the control bits are being sent. If an interjection sequence occurs during an existing interjection (which may only occur if the bus is being “rescued” from some erroneous state), the new interjection **must** present control bits 2'b00.

4.5 Control Bits

The two bits after an interjection are defined as Control Bits. Figure 10 is a flow chart indicating the semantic meaning of the control bits and the resulting state. The first control bit is unconditionally driven by the interjecting layer. If a TX Layer has interjected to indicate the end of transmission, it will put a 1'b1 on the bus for the first control bit. In all other cases the interjector[18]must drive 1'b0 for the first control bit.

If the first control bit was 1'b1 then the addressed receiver (i.e., RX Layer) is responsible for driving the second control bit. If the transmission was sent to the broadcast address (Section 6), all layers—excluding the transmitter (TX Layer) which forwards—should drive the second control bit low to acknowledge. The semantic provided to the transmitter of a broadcast message is either: (1'b1) no layer received the message; or (1'b0) *at least* one layer received the message.

If instead the first control bit was 1'b0 then the interjector is responsible for driving the second control bit. If the interjector is the TX Layer or RX Layer **and** the issue is related to the current transmission, (e.g. RX 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 2'b00 GENERAL ERROR state should be used for general or unclassified interjections as it carries the least semantic meaning.

[16] This shall not be confused with the typical *reset* signal, which usually resets all flip-flops in a corresponding FSM. The MBus reset referred here indicates that the MBus FSM state shall transition to FORWARDING.IDLE state.

[17] While two pulses is sufficient to distinguish an interjection from normal data transmission, >2 pulses provides protection against spurious entry to interjection from a glitch on DATA lines. In the current MBus implementation, six (6) pulses are used for the interjection.

[18] In most of such cases, the interjector would be a layer other than the TX Layer, and this situation indicates an error (either TX or RX error). In the current MBus implementation, the master layer becomes the interjector in such error situations.

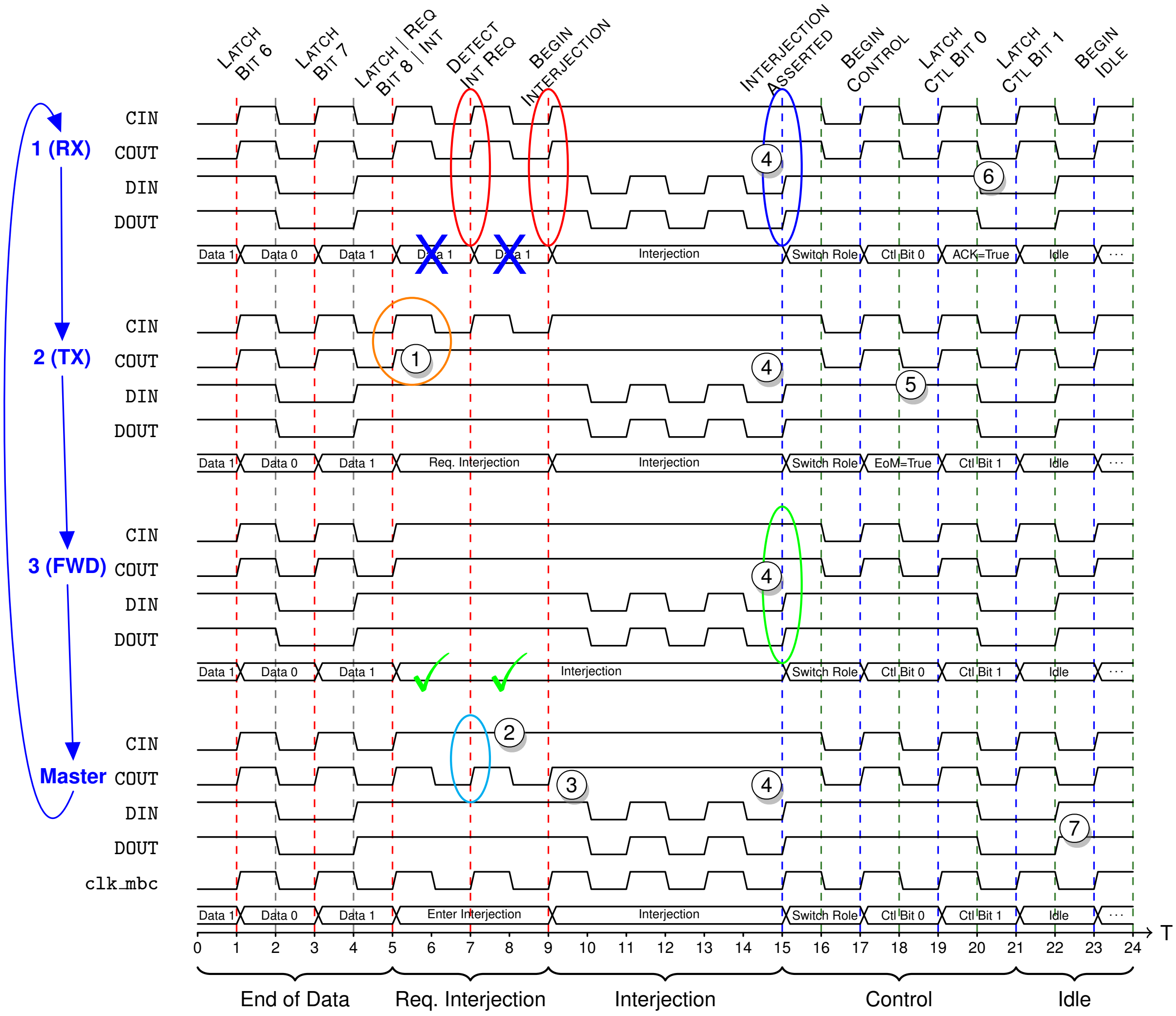


Figure 9: MBus Interjection and Control. MBus interjection sequence provides a reliable in-band reset signal. Any layer may request that the master interject the MBus by holding its COUT high. The master layer detects this and generates an interjection by toggling DATA while holding CLK high. An interjection is always followed by a two-cycle control sequence that defines why the interjection occurred.
This figure shows the interjection timing with successful TX and ACK.
1. After the TX Layer sends all of its data it requests interjection by not forwarding its CLK.
2. Master layer detects that a layer has stopped forwarding CLK.
3. Master layer stops toggling its COUT and begins toggling its DOUT– the interjection sequence.
4. After interjection, the master begins clocking again.
5. The TX Layer signals a complete message by driving Control Bit 0 high.
6. The RX Layer ACK's the message by driving Control Bit 1 low.
7. After control, the master layer stops forwarding DATA, driving it high, and returning the MBus to idle.
For simplicity, this figure shows DATA toggling only 3 times (i.e., 3 rising edges) during interjection. In the current MBus implementation, DATA toggles 6 times (i.e., 6 rising edges) during interjection.

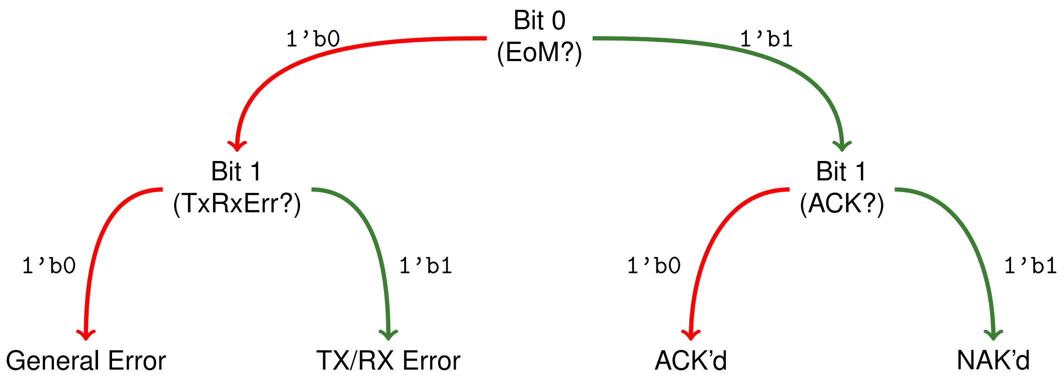


Figure 10: Control Bits. After an interjection, two control bits follow. The first bit is always set by the interjector and indicates whether the interjection was to signal the end of message (EoM). If the EoM bit is high, the RX Layer is responsible for driving the second bit to acknowledge the message. 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 TX Layer or RX Layer encounters an error.

4.6 Return to Idle

After latching the final Control Bit (T=21 in Figure 9), one final edge (T=23) is generated to formally enter `IDLE` state. If, however, the `DATA` line is low at this edge, then it shall be considered the start of a new arbitration cycle instead. The master layer will pull the `CLK` low again in response, which corresponds to T=1 in Figure 8[19].

4.7 MBus Message: Putting It All Together

Figure 11 shows an example of the MBus message in real usage scenario: Enumeration and its reply[20].

Detailed knowledge about the Enumerate and its response is not required yet to understand this figure. However, just to briefly explain this scenario, *Enumerate* is one of the frequently used broadcasting messages. Here, the master layer sends the Enumerate message. Then, one of the member layers should respond to the Enumerate message; here, Layer 1 responds to the message. In this example, the duration of T=3 $\sim$ 91 shows the Enumerate message, and the master layer is the TX Layer, while all the other layers (i.e., Layer 1 and Layer 2) are RX layers[21]. The duration of T=100 $\sim$ 210 shows the Enumerate Response message. For this, Layer 1 is the TX Layer, and the master layer is the RX Layer. Layer 2 is a FWD Layer in this case.

Description on each numbered mark in Figure 11 is given below:

1. Master layer pulls down its `DOUT` to initiate the message transmission. This is forwarded through Layer 1 and Layer 2 and eventually arrives at the master layers' `DIN`. Once the master layer detects that its `DIN` becomes pulled low, it starts operating its internal counter to count "Master Wakeup" period (T=3 $\sim$ 30). Usually, the master layer has been in Active mode before it initiates such MBus messages. Regardless, the current MBus implementation forces the "Master Wakeup" delay for safety, even though there is no actual power-up operation happening during this delay.

2. Once the "Master Wakeup" delay is over, the master layer starts sending out its internal `clk_mbc` through its `COUT`, so that all the layers see and forward the `CLK`. The first three rising edges of `CLK` (T=31, 33, 35)

[19] In the actual MBus implementation, the master layer keeps double-latching its `DIN` once it gets back to `IDLE` state, in order to prevent possible synchronization issue. Thus, the master layer does not immediately pull down its `COUT`. Instead, it pulls down `COUT` 4.5 cycle after the "Begin Idle" edge shown in Figure 9.

[20] Enumerate and Enumerate Response are explained in detail in Section 7.3.

[21] Usually, there is only one RX Layer. However, Enumerate is classified as a *broadcasting* message. By specification, *all* layers must listen to broadcasting messages and respond as needed. Section 7 explains details about the broadcast and non-broadcast messages.

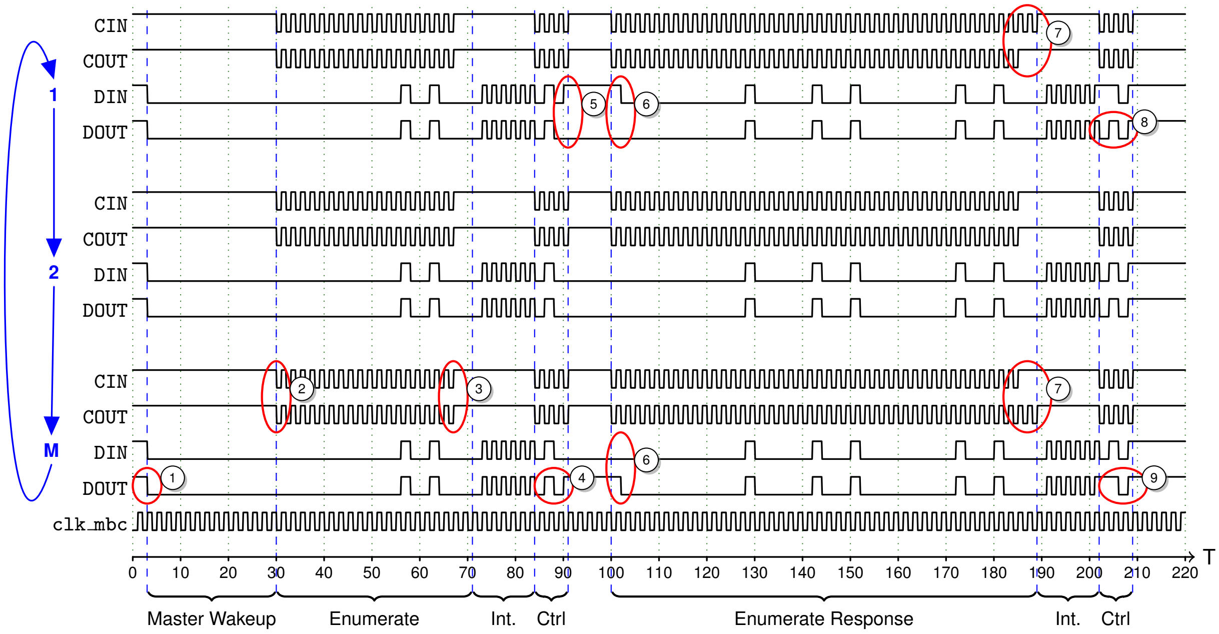


Figure 11: MBus Enumeration and Reply Message '1' and '2' indicate Layer 1 and Layer 2, and they are member layers. 'M' indicates the master layer.

are used for arbitration. From the 4th rising edge (T=37) to 19th rising edge (T=67), it is the TX Layer (the master layer in this example) who drives the DATA line. Other layers just forward the DATA. However, in this example, since Enumerate is a broadcasting message, all layers are forwarding *and* listening to the message. DATA line is latched at rising edge of the CLK line. Hence, this example shows the master layer sending 16'h0024.

3. Once the master layer has sent all the bits (in this case, 16'h0024), it stopped forwarding its clk_mbc to its COUT.[22] Then, the TX Layer (i.e., the master layer) detects that its CIN has stayed high for the set amount of cycles (i.e., 2 cycles), it initiates the interjection. In the current MBus implementation, the interjection consists of six pulses on DATA line.

4. Once the interjection sequence is over, the MBus goes into Control sequence (T=84 $\sim$ 91). The first rising edge of CLK (T=85) is not used.[23]. The second rising edge of CLK (T=87) indicates End-Of-Message (See Figure 10) and driven by TX Layer (i.e., the master layer in this example). Here, the master layer drives its DOUT at 1'b1 since this is the end of the message and it has sent out all the bits successfully. The third rising edge of CLK (T=89) indicates ACK as shown in Figure 10, and RX Layer is responsible for driving the DATA line. For broadcasting messages, all layers, except TX Layer, are RX layers, so in this example, both Layer 1 and Layer 2 are driving the DATA line at 1'b0, which indicates "ACK'd".[24] And this completes an MBus transaction.

5. Although the Enumerate message transaction is over, per MBus protocol, a member layer must respond to the Enumerate message if its internal short prefix address has not been set yet, and in this example,

[22] If the TX Layer were a member layer, it would have stopped forwarding its CIN to COUT. This is shown in the second part of the figure when explaining Enumerate Response.

[23] The first edge of the Control is marked "BEGIN CONTROL" in Figure 9.

[24] All RX layers *must* ACK broadcasting messages. There is no "NAK" for broadcasting messages.

it is Layer 1 who is responsible for sending the reply.[25] Thus, Layer 1 pulls its `DOUT` low (or, holds its `DOUT` low since `DOUT` was already low) at `1'b0` at the last (i.e., 4th) rising edge of `CLK` during the Control sequence, which is at T=91. Basically, this implies the same thing as the master layer pulling its `DOUT` low at T=3. **Every layer pulls its `DOUT` low in order to initiate an MBus transaction**. Note that, at this moment, Layer 1's `DIN` is high. The master layer is in the `IDLE` state since it has completed the previous message transaction (i.e., Enumerate message). Thus it drives its `DOUT` high and this appears at Layer 1's `DIN`.

6. Now the master layer detects its `DIN` becoming low, it starts the arbitration. The master layer always holds its `DOUT` high at the 1st rising edge of `CLK` during arbitration (T=101), unless the master layer is TX Layer. Hence, Layer 1 sees its `DIN` being high at this time. Since its `DIN` is high *and* `DOUT` is low at the 1st rising edge of `CLK`, Layer 1 knows that it has won the arbitration. From T=102, the master layer also starts forwarding the `DATA` line. In this example, no layer tries the priority arbitration, so the `DATA` line is all low at the 2nd rising edge of `CLK` (T=103). Thus, from the 4th rising edge of `CLK` (T=107), Layer 1 becomes the TX Layer. In this example, Layer 1 is sending `40'h0010220044`.
   Note that there is a brief delay before the master layer detects its `DIN` being low (T=91 $\sim$ 100). This is because the master layer keeps double-sampling its `DIN` during `IDLE` state. Hence there is a delay before the master layer detects any transition on the `DATA` line if it is in the `IDLE` state.

7. Once the TX Layer (i.e., Layer 1) has sent all the bits, it stops forwarding the `CLK` line as seen at T=187. However, the master layer keeps toggling its `COUT`. Once the master layer detects that there has been no `CLK` forwarded into its `CIN` for 2 `CLK` cycles, it realizes that the TX Layer has done with the transmission, and it starts the interjection sequence (T=189 $\sim$ 202).

8. Now Layer 1 is the TX Layer, so it is responsible for driving the `DATA` line at the 2nd rising edge of `CLK` (T=205) during the Control. Since it has successfully transmitted all the bits, Layer 1 drives its `DOUT` high at this rising edge, which indicates EoM=`1'b1`.

9. The 3rd rising edge of `CLK` (T=207) in the Control is driven by the RX Layer, which is the master layer in this case. The master layer drives its `DOUT` low to indicate that it has ACK'd the received message. At the 4th rising edge of `CLK` (T=209), no layer tries to initiate another MBus transaction, and the entire `DATA` line is held high. Thus, all layers go into the `IDLE` state and stay there until the next MBus transaction happens.

[25] Details will be explained in Section 7.3.

5 MBus 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.

This section describes the standard power-gating sequence implemented in M^3 systems. The detailed internal signal timings are out of scope of this document and is explained in a separate document[26].

5.1 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 *nearly* zero.

In order to help explain the power-gating sequence in M^3 systems, Figure 12 repeats the previously shown member layer block diagram (in Figure 4.

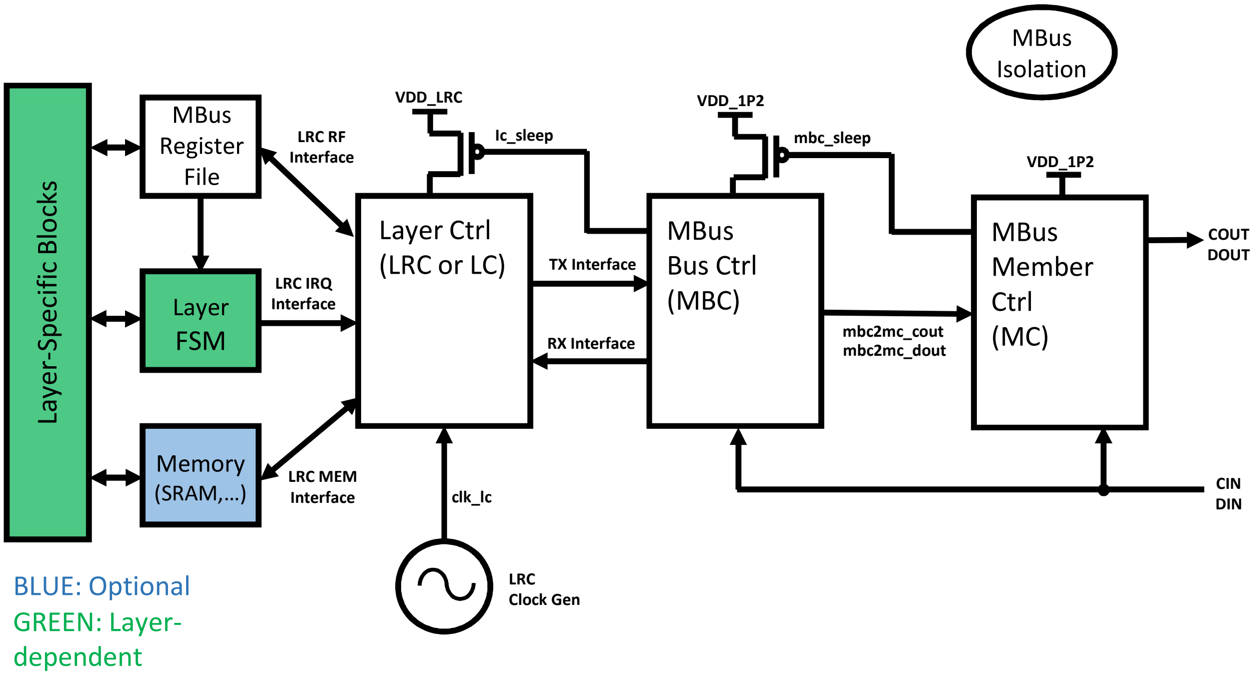


Figure 12: Member layer block diagram

A typical M^3 layer has at least three power domains[27]for MBus implementation: `PD_1P2`, `PD_MBC`, and `PD_LRC`.

`PD_1P2` is an always-on power domain, meaning that it is never powered-off. Its power supply is `VDD_1P2`, as the name suggests. MBus Member Ctrl (MC) and the retentive portion in MBus Register File belong to this power domain. Also, the isolation and level conversion (MBus Isolation) module belongs here. MBus Member Ctrl also generates `mbc_sleep` signal, which turns on/off `PD_MBC` power domain.

`PD_MBC` includes MBus Bus Ctrl, and its power supply is `VDD_1P2` in most M^3 layers. Its header switch is controlled by `mbc_sleep`, which is generated from MBus Member Ctrl. `PD_MBC` becomes *ON*, whenever the layer wakes up (by itself or by an incoming MBus Wakeup message). In most cases, `PD_MBC` becomes *OFF* by an MBus Sleep message.

[26] *MBus Implementation Guide* (pending).
[27] A *Power Domain* is a group of circuitry that share the same power (`VDD`) and ground (`VSS`).

As explained in Section 3.1.1, MBus Bus Ctrl performs the basic decoding and serial-to-parallel conversion for incoming MBus messages. MBus Bus Ctrl can handle all broadcasting messages (Section 7.3) without activating other power domains. However, if the incoming MBus messages are targeted to this layer — *AND* — the MBus message is one of the Register (Section 7.5.2 or Section 7.5.3) or the Memory (Section 7.6.1 or Section 7.6.2 or Section 7.6.3) messages, then MBus Bus Ctrl turns on `PD_LRC` power domain and delivers the received MBus messages to Layer Ctrl through the RX Interface. In order for this, MBus Bus Ctrl generates `lc_sleep` signal, which turns on/off `PD_LRC` power domain.

`PD_LRC` includes Layer Ctrl (LRC). Its power supply is `VDD_1P2` in most M^3 layers, but some layers, such as Processor (PRC), uses `VDD_0P6` to reduce active power consumption. If the layer has a memory (e.g., SRAM), then usually the memory and its controller are also in `PD_LRC`, since Layer Ctrl has the direct interface with the memory through LRC MEM Interface[28]. In most cases, Layer FSM or Layer-Specific Blocks also tend to reside in `PD_LRC`, since this simplifies the overall power domain structure in M^3 layers. However, high-power analog blocks (if any) tend to have their own power domains. In this case, these additional power domains are not considered as part of the MBus specification, and are purely implementation-defined.

Table 3: M^3 Layer Standard Power Domains

Power Domain	Power Supply	Turned On By	Turned Off By	Modules
`PD_1P2`	`VDD_1P2`	Always	Never	MBus Register File[29] MBus Master Ctrl MBus Member Ctrl
`PD_MBC`	`VDD_1P2`	Self-Wakeup or MBus Wakeup Msg	MBus Sleep Msg	MBus Bus Ctrl
`PD_LRC`	`VDD_1P2` or `VDD_0P6`	MBus Bus Ctrl	MBus Sleep Msg	Layer Ctrl Other Modules (if any)

5.2 Self-Wakeup

In actual M^3 systems, the very first wakeup is always a self-wakeup, meaning that one of the always-on blocks in a layer interrupts the always-on MBus Master Ctrl or MBus Member Ctrl to send out the MBus *NULL* transaction. It could be an external programming interface (such as GOC or EP interface in PRC—See PRC documentation) or a wakeup timer in the layer who interrupts the MBus Master Ctrl or MBus Member Ctrl. In any case, the result is the MBus NULL message generated by the MBus Master Ctrl or MBus Member Ctrl. Figure 13 shows the waveform of the MBus NULL message.

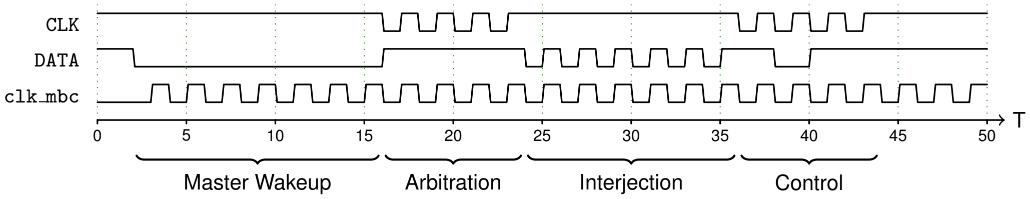


Figure 13: MBus NULL (Self-Wakeup) Message

Note that `clk_mbc` was not running before `DATA` being pulled low (T=2), indicating that (at least) the master layer was in Sleep mode. Any layer can pull down its `DOUT` low to initiate the system wakeup. It is usually one

[28] MBus Register File also has the direct interface with Layer Ctrl through LRC RF Interface. In most cases, MBus Register File is in the always-on power domain (`PD_1P2`), so Layer Ctrl can always access MBus Register File.

[29] There are various types of registers in MBus Register File. Although, traditionally, most of registers in MBus Register File are in `PD_1P2` power domain, it is possible to have registers in `PD_LRC` or their own power domains. Section 9 explains the details.

of the always-on blocks in such layer that interrupts its MBus Master Ctrl or MBus Member Ctrl to pull down its DOUT. Also note that, at this point, no other MBus-related blocks, other than the always-on MBus Master Ctrl and MBus Member Ctrl, are in active. This means that, no layer can send any data at this point; hence, *NULL* message.

Master layer turns on MBus Bus Ctrl (PD_MBC) as soon as it detects its DIN falling (T=2), and this starts clk_mbc running. After some delay (T=2 ∼ 16) reserved for the master layer wakeup, the master layer starts toggling the CLK line. As explained earlier, the first three rising edges of CLK (T=17, 19, 21) are used for arbitration, and the master layer holds DATA high at the first edge of CLK (T=17). Since no other layer participates in the arbitration, the master layer detects no winner, and immediately starts the interjection sequence (T=24). This situation is regarded as a TX/RX error, so the master layer drives the two Control bits at 2'b01 (See Figure 10). And then, the master layer goes back to IDLE state.

The result of this NULL message is:

- Master Layer
  - Turns on MBus Bus Ctrl (PD_MBC) at the first falling edge of DIN (T=2), and immediately starts running clk_mbc.
  - Turns on Layer Ctrl (PD_LRC) at the first falling edge of CLK (T=16).
- Member Layer
  - Turns on MBus Bus Ctrl (PD_MBC) at the first rising edge of CLK (T=17).

Note that, Layer Ctrl (PD_LRC) in any member layer does **NOT** turn on by the NULL message by default.

Since the NULL message effectively wakes up all MBus blocks, except Layer Ctrl in member layers, it is also called *Self-Wakeup* message.

5.3 Sleep

Any layer that tries to put the entire system into Sleep mode shall send the MBus All-Sleep message. This MBus All-Sleep message is defined as one of the broadcasting messages (See Section 7.3), so detailed explanation is deferred for now. For completeness, below Figure 14 shows the MBus All-Sleep message[30]. The transmitted bit sequence is 16'h0100.

Note that, at the end of the MBus All-Sleep message, clk_mbc stops running (T=94). This implies that the master layer has gone to Sleep mode. Other member layers also go into Sleep mode at the end of the MBus All-Sleep message.

Here, “Sleep mode” means that all MBus-related power domains become turned off (i.e., PD_MBC and PD_LRC).

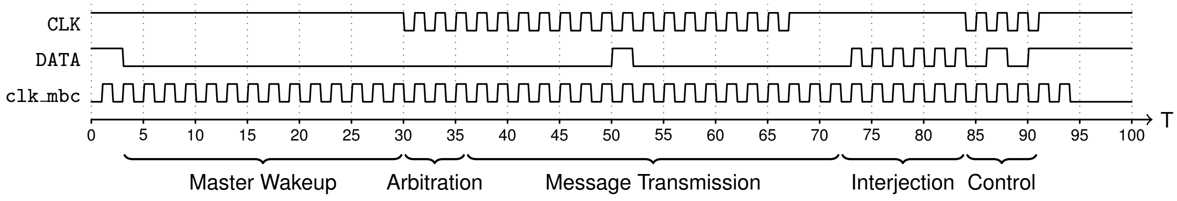


Figure 14: MBus All-Sleep Message

[30] The MBus All-Sleep message is often called just “MBus Sleep message” for brevity.

6 MBus Address Design

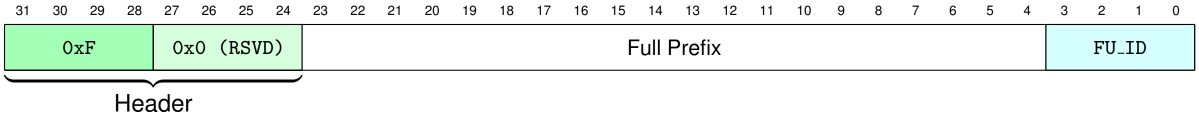
One of the MBus design points is to serve as a system interconnect for a physically constrained system. With that in mind, MBus attempts to optimize for a system of complex connected components. In particular, MBus expects that an individual member layer may itself be composed of multiple *functional units*, which may each be individually addressed. In the current MBus implementation, the functional units include MBus Register File and Memory (if any), and the functional unit identifier (`FU_ID`) is assigned separately for read and write operation.

6.1 Address Types

MBus defines the term *prefix* to refer to the portion of the address that specifies which layer is being addressed and reserves the term *address* to refer to a complete address —one that specifies a functional unit within a layer. Thus, 'address' is 'prefix' + 'functional unit identifier (`FU_ID`)'.

Every MBus layer 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 `FU_ID`. A full address is the composition of a header, a full prefix, and a `FU_ID`. To distinguish full and short addresses, the short prefix `0xF` is reserved to identify a full address. The first four bits of the full address header is thus always `0xF`.

6.2 Full Prefix

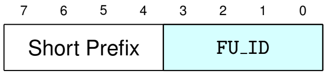


The purpose of full prefixes is to serve as a unique identifier for a layer, akin to a product identifier. Full prefixes do not distinguish instantiations of a layer, that is, multiple copies of a unique layer will all have the same full prefix. This implies that multiple layers in a single M^3 system may have the same full prefix. Bits 24-27 of the full address are reserved. The remaining range, bits 4-23, are available to be utilized as full prefixes.

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

Full prefixes for layers used in M^3 systems are currently managed by the University of Michigan (*the University*) and CubeWorks, Inc. (*CubeWorks*). Designers can assign any full prefix for new layers, but it may not be considered an certified MBus implementation unless approved by the University of Michigan or CubeWorks.

6.3 Short Prefix



The purpose of short prefixes is to uniquely identify layers in an M^3 system. Multiple layers in an M^3 system **must not** have the same short prefix.

- The short prefix `0x0` is reserved as the broadcast prefix.
- The short prefix `0x1` is reserved for master layer.
- The short prefix `0xF` is reserved to distinguish full addresses.

This leaves a remainder of 14 unique short prefixes. These short prefixes map to actual layers instantiated in an M^3 system. If there are multiple copies of the same layer type (e.g. several external memory chips), each instance is given a unique short prefix.

In an M^3 system, short prefixes are assigned dynamically. Out of reset, all member layers have the default short prefix, 0xF, which implies “Not-Yet-Assigned”. After system powers on, some layer shall send a series of Enumerate messages (See Section 7.3) to *enumerate* (i.e., assign a short prefix) each layer.

Usually, it is the master layer who sends the Enumerate messages. The master layer itself does not need to be enumerated (i.e., assigned a short prefix), since its short prefix is hard-wired to 0x1.

If the M^3 system has N member layers, the master layer must send out N Enumerate messages, each with unique short prefix. The short prefix assignments are resolved by the topological priority arbitration; the first Enumerate message enumerates the layer whose DIN is connected to the master layer’s DOUT, and any subsequent Enumerate message enumerates the subsequent layer (i.e., the layer whose DIN is connected to the previously enumerated layer’s DOUT).

It is the responsibility of the layer performing enumeration to ensure no duplicate short prefixes are assigned.

Since three of the possible sixteen short prefixes are reserved (0x0, 0x1, 0xF), an M^3 system can have up to 14 layers (1 master layer + 13 member layers), unless it utilizes the full prefix.[31]

Short prefixes cannot be used to affect priority. Priority is determined exclusively by physical topology.

Short prefix assignments **must** be preserved when layers are power-gated. Thus, it is MBus Master Ctrl or MBus Member Ctrl, which is in the always-on power domain (PD_1P2), that has a register to store the layer’s short prefix.

[31] The current MBus implementation is not capable of handling full prefixes, and will ignore any MBus addresses starting with 0xF. Thus, the 14 layers (1 master layer + 13 member layers) is the maximum number of layers that an M^3 system can have.

7 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.

7.1 Bitfield Representation

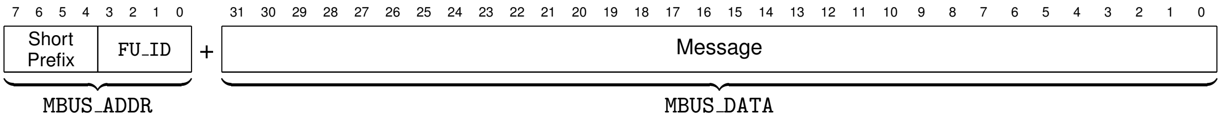
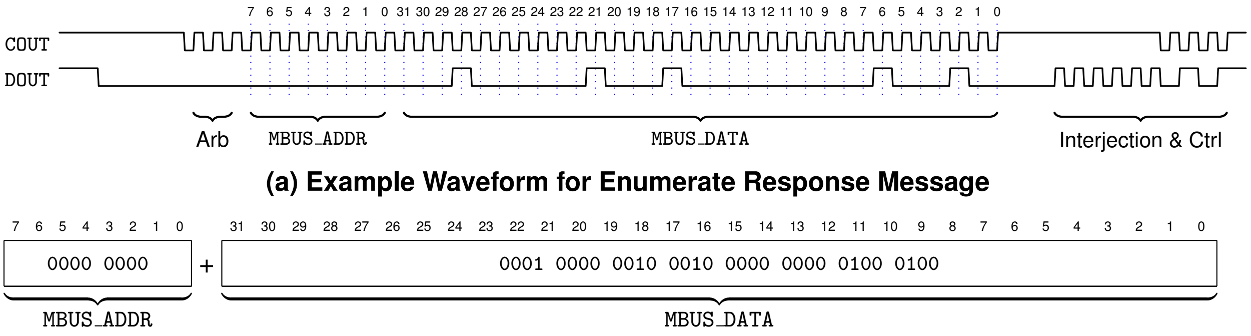


Figure 15: MBUS_ADDR and MBUS_DATA in a General MBus Message MBUS_DATA section may be repeated if the amount of Data to be sent is $\geq$ 32 bits.



(a) Example Waveform for Enumerate Response Message

(b) Corresponding Bitfield Representation

Figure 16: MBus Message Waveform and Its Bitfield Representation Both represent an MBus message with MBUS_ADDR =0x00 and MBUS_DATA =0x10220044. Arbitration, Interjection, and Control bits are not included in bitfield representations.

As shown in Figure 15, bitfields are used to present 8-bit Address + 32-bit Data. The 8-bit-Address is called MBUS_ADDR, and the 32-bit-Data is called MBUS_DATA.

If the full addresses were used, it would have been 32-bit MBUS_ADDR and 32-bit MBUS_DATA. In this document, all MBUS_ADDR is assumed to be a *Short Address*, unless otherwise explicitly specified.

MBUS_ADDR is broken down into the 4-bit *Short Prefix* and the 4-bit *Functional Unit ID (FU_ID)*. If the full addresses were used, the 32-bit MBUS_ADDR would have looked like what was shown in Section 6.2.

MBUS_DATA shown here is 32-bit long. However, the only requirement is that it should be a multiple of one byte (8 bits). In the current MBus implementation, the length of MBUS_DATA is one of the following:

- 0 bits: NULL message[32](Figure 13)
- 8, 24, 32 bits: Broadcast messages
- $N\times$ 32 bits: Register or Memory messages

[32] NULL message does not even have MBUS_ADDR.

The bitfields representation directly mimics the actual MBus waveform, although it does not include bit sequences for Arbitration, Interjection, and Control bits. For example, the MBus All-Sleep message waveform shown in Figure 14 shows that the transmitted bit sequence is 16'h0100[33]. Here, the first 8 bits are MBUS_ADDR (because it uses the short address), and the remaining is MBUS_DATA. Thus, it is equivalent to saying that MBUS_ADDR = 8'h00 and MBUS_DATA = 8'h00. Similarly, the Enumerate Response message shown in Figure 11 has MBUS_ADDR = 8'h00 and MBUS_DATA = 32'h10220044.

Figure 16 compares the actual waveform of the Enumerate Response and its bitfield representation.

In the bitfields, 1'b0 and 1'b1 indicate bits that must be set to that value, 1'bX indicates bits that depend on the current message, and 1'bZ indicates bits that should be *ignored*—accept any value, send as 1'b0. *Insignificant Bits* are also indicated as 1'bZ.

7.2 Decoding MBus Address (MBUS_ADDR)

Table 4 shows possible MBUS_ADDR values and there meaning.

MBUS_ADDR[7:4]=4'h0 indicates broadcast messages. In this case, MBUS_ADDR[3:0] indicates broadcast channel ID. Broadcast messages are described in Section 7.3

MBUS_ADDR[7:4]=4'hF is reserved for use with full prefix. Currently it is not implemented in the current MBus implementation.

MBUS_ADDR[7:4]=4'h1∼4'hE represents a short prefix. A layer assigned this value as its short prefix must listen to this message and respond as needed[34]. In this case, MBUS_ADDR[3:0] indicates FU_ID.

Details of each functions are explained in following sections. See the related sections shown in the table.

Table 4: MBUS_ADDR

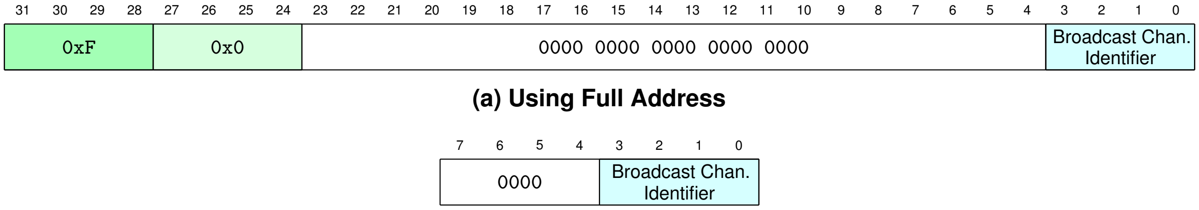
MBUS_ADDR					Description	Related Section
[7:4]	[3]	[2]	[1]	[0]		
4'h0	0	0	0	0	Layer Discovery and Enumeration	Section 7.3.2
	0	0	0	1	Layer Power	Section 7.3.3
	4'h2–4'hF				Reserved	-
Short Prefix (4'h1 ∼4'hE)	0	0	0	0	Register Write	Section 7.5.2
	0	0	0	1	Register Read	Section 7.5.3
	0	0	1	0	Memory Bulk Write	Section 7.6.1
	0	0	1	1	Memory Read	Section 7.6.3
	0	1	0	0	Memory Stream Write (Channel 0)	Section 7.6.2
	0	1	0	1	Memory Stream Write (Channel 1)	
	0	1	1	0	Memory Stream Write (Channel 2)	
	0	1	1	1	Memory Stream Write (Channel 3)	
	1	X	X	X	Reserved	-
4'hF	X	X	X	X	Reserved (Full Prefix)	-

7.3 Broadcast Messages (Address 0x0X, 0xf000000X)

MBus defines the broadcast short prefix as 4'h0 and the broadcast full prefix as 20'h00000. Broadcast messages are permitted to be of arbitrary length. Messages longer than 32 bits may be silently dropped by layers with small buffers. A layer **must not** interject a broadcast message to indicate buffer overflow. Interjections are permitted for broadcast messages greater than 4 bytes in length.

[33] Note that the first three rising edges of CLK in the waveform are for arbitration. These are not counted as the transmitted data bits.
[34] The message may be silently ignored if there is no layer with the specified short prefix.

For broadcast messages, the `FU_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.



(a) Using Full Address

(b) Using Short Address

Figure 17: MBus Addresses Used for Broadcasting Messages

The MSB of the broadcast channel identifier (address bit 3) shall identify MBus broadcast operations. If the MSB is `1'b0`, it indicates an official MBus broadcast message as specified in this document and subsequent revisions. Broadcast messages with a channel MSB of `1'b1` are implementation-defined.

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

7.3.1 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 layer 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 the reserved channels, follow a common template. The messages consist of a 8-bit- or 32-bit-long `MBUS_ADDR` (depending on the use of short/full addresses) and a 32-bit-long `MBUS_DATA`. The four most significant bits in `MBUS_DATA` identify the message type/command. Some messages do not require all 32-bit-long `MBUS_DATA`. The unused bits are named *insignificant bits*. `MBUS_DATA` section may be truncated, omitting the insignificant bits on the wire[35].

All examples are shown with short addresses for space. There is no distinction between the use of the short or full broadcast address.

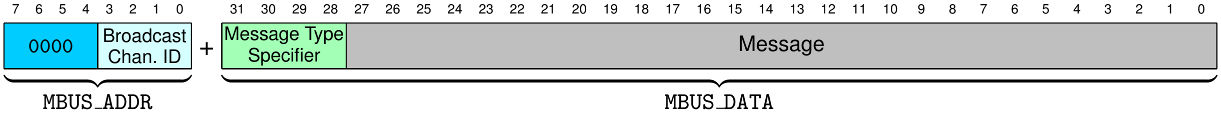


Figure 18: `MBUS_ADDR` and `MBUS_DATA` in an MBus Broadcasting Message

`MBUS_ADDR` is broken down into the 4-bit *Broadcast Prefix* and the 4-bit *Broadcast Channel ID*, as shown in Figure 17b. As noted earlier, the 4-bit Broadcast Prefix must be `4'b0000`.

`MBUS_DATA` is broken down into a 4-bit *Message Type Specifier* and a 28-bit Message.

7.3.2 Broadcast Channel 0: Layer Discovery and Enumeration

Channel 0 is used for messages related to layer 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 MBus Bus Ctrl is responsible for handling Channel 0 messages.

[35] 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

7 6 5 4	3 2 1 0		31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0000	0000	+	0000	ZZZZ	ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ

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

Query/Enumerate Response

7 6 5 4	3 2 1 0		31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4	3 2 1 0
0000	0000	+	0001	ZZZZ	Full Prefix	Short Prefix

This message is sent in response to a Query, Enumerate, or Invalidate Short Prefix request.

When responding to Query, every layer will be transmitting their address, and layers should anticipate losing arbitration several times before they are able to send their response. Layers **must** retry until the message is sent.

When responding to Enumerate, layers **must not** retry sending if arbitration is lost and **must** retry sending if interjected[36].

The top four bits of `MBUS_DATA` identify the message as a Query/Enumerate Response. The next four bits are ignored. The following 20 bits contain the full prefix of the layer. The final 4 bits are the currently assigned short prefix. Layers that have not been enumerated should report a short prefix of `0xF`.

Enumerate

7 6 5 4	3 2 1 0		31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0000	0000	+	0010	Short Prefix	ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ

This message assigns a short prefix to a layer. All layers that receive this message and do not have an assigned short prefix **must** attempt to reply with a Query/Enumerate Response. Layers with a short prefix shall ignore the message (broadcast NAK, that is, forward). Layers shall perform exactly one attempt to reply to this message. The layer that wins arbitration shall be assigned the short prefix from this message. Layers that lose arbitration shall remain unchanged.

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

Invalidate Short Prefix

7 6 5 4	3 2 1 0		31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0000	0000	+	0011	Short Prefix	ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ

This message clears the assignment of a short prefix. The bottom 4 bits specify the layer whose prefix shall be reset. A layer shall reset its short prefix to `0xF`. If the short prefix to clear is set to `0xF` in the message, then all layers shall reset their short prefixes.

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

7.3.3 Broadcast Channel 1: Layer Power

Channel 1 is used to query and command the Layer power state[37]. All layers capable of entering a low-power state **must** enter their lowest power state in response to an All Sleep message.

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

All Sleep

7	6	5	4	3	2	1	0		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0000				0001				+	0000				ZZZZ				ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ																						

All layers 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 Wake

7	6	5	4	3	2	1	0		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0000				0001				+	0001				ZZZZ				ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZ																						

All layers receiving this message **must** immediately wake up, that is, they have to turn on `PD_MBC` and `PD_LRC`. The bottom 28 bits of this message are reserved and should be *ignored*.

Selective Sleep By Short Prefix

7	6	5	4	3	2	1	0		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0000				0001				+	0010				Z	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Z	ZZZZ				ZZZZ ZZZZ							

This message instructs selected layers to sleep. The 16 bits of data are treated as a bit vector, mapping short prefixes to bit indices. That is, the layer with short prefix `4'b1101` is controlled by the second bit received (bit 25 in the bit vector above). If a bit is set to `1'b1`, the selected layer **must** enter Sleep mode. If a bit is set to `1'b0`, the selected layer should not change power state. The bits for prefixes `4'b1111` and `4'b0000` are ignored.

Selective Wake By Short Prefix

7	6	5	4	3	2	1	0		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0000				0001				+	0011				Z	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Z	ZZZZ				ZZZZ ZZZZ							

This message instructs selected layers to wake. The 16 bits of data are treated as a bit vector, mapping short prefixes to bit indices. That is, the node with short prefix `4'b1101` is controlled by the second bit received (bit 25 in the bit vector above). If a bit is set to `1'b1`, the selected layer **must** wake up. If a bit is set to `1'b0`, the selected layer should not change power state. The bits for prefixes `4'b1111` and `4'b0000` are ignored.

[37] Power-oblivious layers may ignore channel 1.

Selective Sleep By Full Prefix

7 6 5 4	3 2 1 0		31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4	3 2 1 0
0000	0001	+	0100	ZZZZ	Full Prefix	ZZZZ

This message instructs selected layers to sleep. Any layer whose full prefix matches **must** enter sleep.

Selective Wake By Full Prefix

7 6 5 4	3 2 1 0		31 30 29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4	3 2 1 0
0000	0001	+	0101	ZZZZ	Full Prefix	ZZZZ

This message instructs selected layers to wake. Any layer whose full prefix matches **must** wake up.

7.3.4 Broadcast Channel 2–15: Reserved

Broadcast channels 2–15 are reserved.

7.4 Extension Messages (Address `0xffffffff`)

This address is reserved for future extension to MBus.

7.5 Register Messages

7.5.1 Brief Intro to MBus Register File

MBus Register File is one of the main components in MBus implementation as shown in Figure 3 and Figure 4.

MBus Register File is a group of registers. Each register can have up to 24 bits. There can be un-used (i.e., empty) bits in the middle of a register, or it is permitted to have an entirely empty register (i.e., all 24 bits are not used) Un-used bits are treated as RZWI (Read as Zero, Write Ignored).

MBus Register File can have up to 256 registers. Hence, the maximum number of bits that MBus Register File can have is 256 registers × 24 bits/register = 6144 bits.

Each register is assigned its index, Register ID (`REG_ID`), starting from 0 and incrementing by 1. Hence, if an MBus Register File has 256 registers, the `REG_ID` spans from 0 to 255. Obviously, 8 bits are needed to represent/store `REG_ID`.

It is not required to have all the 256 registers in MBus Register File. In fact, most M^3 layers have far less than 256 registers in MBus Register File. However, it is required that Layer Ctrl know the `REG_ID` of the last register in MBus Register File at design time; it is hard-wired and cannot be changed once fabricated.

In this document, the `REG_ID` of the last register will be referred to $\mathtt{N}_{\mathtt{R}}$.[38]

Usually, MBus Register File is in the always-on `PD_1P2` power domain, hence preserving its data even in Sleep mode. However, it is also possible to have *non-retentive* registers or registers in a completely separate power domain. This will be explained in Section 9.

[38] This shall not be confused with the actual number of registers in MBus Register File. There may be empty registers, in which case, the actual number of registers is less than $\mathtt{N}_{\mathtt{R}}$. In fact, the actual number of registers is **always** equal to or less than ($\mathtt{N}_{\mathtt{R}} + 1$).

It is important to note that, only Layer Ctrl has a *write access* to MBus Register File. If there is any other module or block that needs to write into MBus Register File, it has to use the LRC Interrupt Interface (Section 8.1).

On the contrary, all modules and blocks in a layer can have a *read access* to MBus Register File. They can simply use wires coming out from desired register outputs; there is no addressing involved, and all the outputs are visible. This also means that, the read access to MBus Register File is determined at design time; it is hard-wired and cannot be changed once fabricated. By default, Layer Ctrl has a read access to *all* registers in MBus Register File.

The last 64 registers in MBus Register File can be used as MBus Support Registers. Details on MBus Support Registers will be explained in later sections when discussing MBus Register and Memory messages.

7.5.2 Register Write Message

7 6 5 4	3 2 1 0		31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
. . . Prefix	0000	+	REG_ID	REG_DATA	REG_ID . . .
MBUS_ADDR			MBUS_DATA		

REG_DATA (MBUS_DATA[23:0]) are written to the register indexed by REG_ID (MBUS_DATA[31:24]). The write occurs immediately, as soon as the Layer Ctrl receives the message. Multiple registers may be written in a single MBus transaction by sending multiple MBUS_DATA sections. Each 32 bit chunk of MBUS_DATA is treated as if it were an independent transaction.

Status Registers A part of MBus Support Registers is used for status registers. Upon writing into the corresponding register(s), Layer Ctrl updates the following MBus Support Registers.

	23 22 21 20	19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
Reg#(N				

7.5.3 Register Read Message

7 6 5 4	3 2 1 0		31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
. . . Prefix	0001	+	First REG_ID to Read From	NUM_REGS-1	Target MBUS_ADDR	Target REG_ID on Destination
MBUS_ADDR			MBUS_DATA			

- First REG_ID to Read From (MBUS_DATA[31:24]) specifies the index of the first register to be read.
- NUM_REGS-1 (MBUS_DATA[23:16]) is used to request that multiple registers are sent. This field is a count of the number of registers to be sent less one. For example, if MBUS_DATA[31:24]=5 and MBUS_DATA[23:16]=0, then only Reg#5 is read and sent. If MBUS_DATA[31:24]=5 and MBUS_DATA[23:16]=2, then Reg#5 ~ #7 (total 3 registers) are read and sent.
- Target MBUS_ADDR (MBUS_DATA[15:8]) are the MBUS_ADDR to be used in the resulting response message.
- Target REG_ID on Destination (MBUS_DATA[7:0]) specifies the first REG_ID field in the resulting response message.

The response message has Target MBUS_ADDR (MBUS_DATA[15:8]) in its MBUS_ADDR field, and its MBUS_DATA field is formatted exactly as the Register Write Message: 8 bit REG_ID + 24 bit REG_DATA. For reads of more than one register, the REG_ID field in the response is incremented by 1 for each register.

Figure 19 shows an example of a register read transaction and shows how the response message is made.

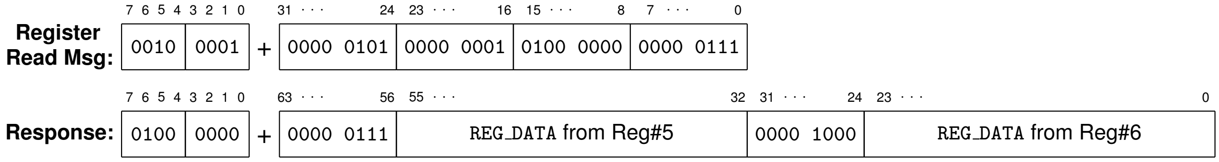


Figure 19: Example of MBus Register Read transaction

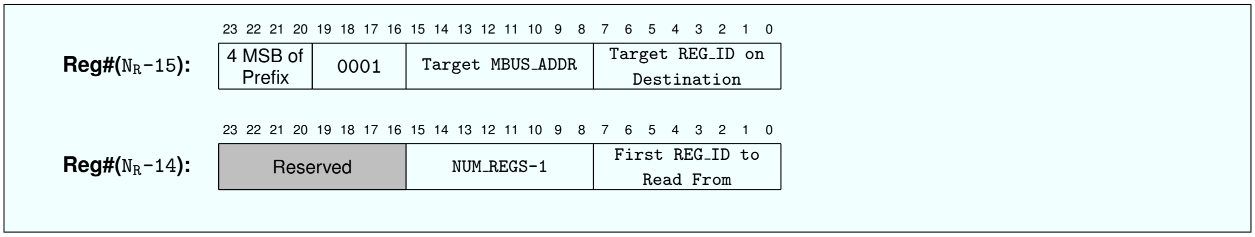
Register Read Msg: `MBUS_ADDR[7:4]` specify that this message is targeted to a layer whose short prefix is `0x2`. `MBUS_ADDR[3:0]` is `FU_ID` and the value `0x1` means this is a Register Read message. `MBUS_DATA[31:24]` specify that it should read from Reg#5 in the target layer (i.e., the layer whose short prefix is `0x2`). `MBUS_DATA[23:16]` specify that it should read 2 registers, which implies that it should read Reg#5 and Reg#6 in this example. `MBUS_DATA[15:8]` specify the `MBUS_ADDR` to be used in the response message. `MBUS_DATA[7:0]` becomes the first `REG_ID` field in the response message. Upon receipt of this 'Register Read Msg', the target layer (i.e., the layer whose short prefix is `0x2`), should generate the message shown in 'Response'.

Response: `MBUS_ADDR` is directly copied from `MBUS_DATA[15:8]` in 'Register Read Msg'. `MBUS_DATA[63:56]` is directly copied from `MBUS_DATA[7:0]` in 'Register Read Msg'. `MBUS_DATA[55:32]` is `REG_DATA` from Reg#5 in the layer whose short prefix is `0x2`. `MBUS_DATA[31:24]` is the previous `REG_ID` (i.e., `MBUS_DATA[63:56]`) incremented by 1. `MBUS_DATA[23:0]` is `REG_DATA` from Reg#6 in the layer whose short prefix is `0x2`.

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

Note: The `Target MBUS_ADDR` **must** be copied exactly. The `FU_ID` is not required to be as shown in Register Write Message.

Status Registers A part of MBus Support Registers is used for status registers. After sending the complete response message, Layer Ctrl updates the following MBus Support Registers.

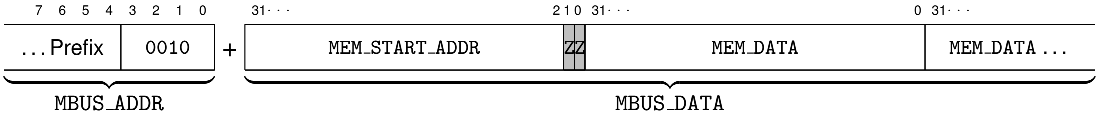


7.6 Memory Messages

MBus 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). MBus provides two types of memory commands: **bulk** and **stream**.

- A bulk memory transaction is a wholly self-contained event which includes all necessary information required for memory write (i.e., `MEM_ADDR` and `MEM_DATA`).
- Memory streams pre-configure the stream information in MBus Support Registers on the destination layer (RX Layer) and omit it from subsequent transactions, relying on the RX Layer to maintain and increment a destination pointer. Streams are useful for applications such as continuous sampling, where multiple, short messages are generated.

7.6.1 Memory Bulk Write Message



MEM_START_ADDR is the address in memory to begin writing to. The bottom two bits of this field are reserved and **must** be transmitted as 0.

Subsequent words in MBUS_DATA are treated as MEM_DATA to be written. The first MEM_DATA is written to MEM_START_ADDR. The next MEM_DATA is written to MEM_START_ADDR +4 (the next word in memory) and so on. The length of this message is usually limited by the master layer. In PRCv20 Family, the limit is 8 bits + 64kB by default,[39]which means, aside from the 8-bit MBUS_ADDR, MBUS_DATA could be as long as 64kB, which would be further divided down to 32-bit MEM_START_ADDR + up to 16383 words in MEM_DATA.

Configuration Registers A part of MBus Support Registers is used for configuration registers as shown below.

	23	22	21	20	19	18	17	16	15 … 0
Reg#(N_R-39):	EN	CACT	RSV	RSV	RSV	RSV	RSV	RSV	Length Limit-1

- Enable (EN)
  - Controls whether bulk memory transactions written to this layer are enabled. If EN is 1'b0, a layer must not modify the contents of memory in response to a bulk memory transaction.
- 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 RX Layer should interject with RX error as soon as it knows the length limit has been exceeded.
- Length Limit−1
  - The maximum permitted length in words of a memory bulk write message.

Status Registers A part of MBus Support Registers is used for status registers. When the write completes, Layer Ctrl updates the status registers as shown below.

	23 … 20	19 … 16	15 … 2	1	0
Reg#(N_R-15):	4 MSB of Prefix	0010	MEM_START_ADDR[15:2]	RSV	RSV

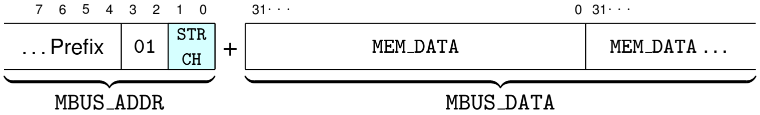
	23 … 16	15 … 0
Reg#(N_R-14):	Reserved	MEM_START_ADDR[31:16]

	23 … 16	15 … 0
Reg#(N_R-11):	Reserved	NUM_WORDS-1

[39] This value can be changed by user. The absolute limit in PRCv20 Family is 1048575 bits (8-bit MBUS_ADDR + up to 1048567-bit MBUS_DATA). Thus, Memory Bulk Write message can contain up to 32766 words of MEM_DATA, and Memory Stream Write message can contain up to 32767 words of MEM_DATA.

Overflow If more data is received after writing the last address in the 32-bit address space (`0xFFFFFFFC`), the destination address wraps and data continues to be written, beginning at address `0x00000000`.

7.6.2 Memory Stream Write Message

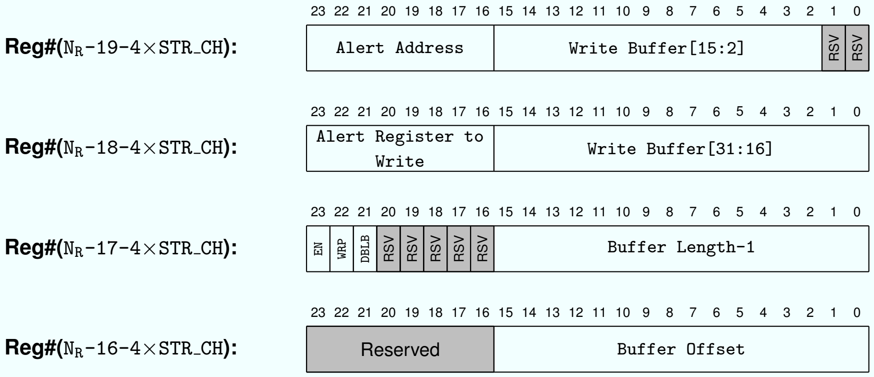


MBus layers have up to four streaming memory channels (`STR_CH`). Each channel is controlled by its configuration registers in MBus Support Registers. The destination of a memory stream write is specified by a combination of channel selection—the last two bits of the `FU_ID`— and the pre-arranged configuration. The following paragraph Configuration Registers describes more details.

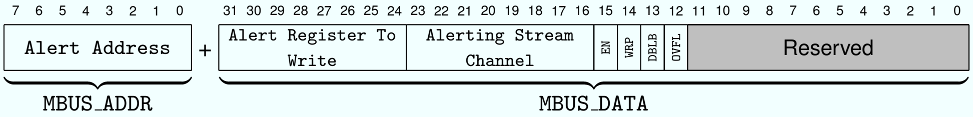
The message payload is all data. The destination address is automatically incremented every time a word is written.

The length of this message is usually limited by the master layer. In PRCv20 Family, the limit is 8 bits + 64kB by default,[39]which means, aside from the 8-bit `MBUS_ADDR`, `MBUS_DATA` could be as long as 64kB (i.e., up to 16384 words in `MEM_DATA`).

Configuration Registers A part of MBus Support Registers is used for configuration registers as shown below. Note that, each streaming channel (`STR_CH`) has its own set of configuration registers, and this is implied in the `REG_ID` s shown below. For example, `STR_CH` =0 uses Reg#(N_R-19) ∼ Reg#(N_R-16) for configuration registers, while `STR_CH` =3 uses Reg#(N_R-31) ∼ Reg#(N_R-28).



- `Alert Address`
  - Defines where alerts are sent. If the alert prefix (`Alert Address[7:4]`) are set to the full-prefix indicator `4'b1111`, the alert is suppressed.
  - Alerts are sent whenever the end of the buffer is reached. If `DBLB` is `1'b1`, an alert is also sent when the halfway point of the buffer is reached.
  - When a memory stream alert occurs, a layer sends the following message:



    - `MBUS_ADDR` is set to the `Alert Address`.
    - `MBUS_DATA[31:24]` are set to the `Alert Register to Write`.

        * MBUS_DATA[23:16] are the Alerting Stream Channel, made up of this layer's short prefix, followed by 2'b01, followed by the stream channel (STR_CH) that generated the alert —this should be the same address used to write to this stream channel.
        * MBUS_DATA[15] is the EN bit, which reports the current state of the EN bit of the alerting channel when the alert was sent.
        * MBUS_DATA[14] is the WRP bit, which is set if the write transaction that generated this alert reached the end of the stream buffer.
        * MBUS_DATA[13] is the DBLB bit, which is set if the write transaction that generated this alert reached the halfway point of the stream buffer and double-buffering is active for this stream.
        * MBUS_DATA[12] is the OVFL bit, which is set if the write transaction 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 transaction 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.

- Write Buffer[31:0]
    - Pointer to the beginning of a buffer in memory. Write Buffer[1:0] is always 2'b00.
- Buffer Offset[15:0]
    - Buffer offset used to determine the memory address for write.
- Buffer Length−1
    - Defines the length of the buffer.
- Enable (EN)
    - Controls whether this channel is enabled. If EN is 1'b0, a layer must not modify memory in response to a memory stream message.
- Wrap (WRP)
    - Defines layer behavior when the end of the buffer is reached. If WRP is 1'b1, the Write Address Counter should reset to its original value. If WRP is 1'b0, 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 1'b0.
- Double Buffer (DBLB)
    - Controls double-buffering mode. If double-buffering is active (DBLB=1'b1), the layer 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.

The destination memory address is determined as below:

MEM_ADDR = Write Buffer + (Buffer Offset << 2)

where Write Buffer and Buffer Offset are read from corresponding channel's configuration registers.

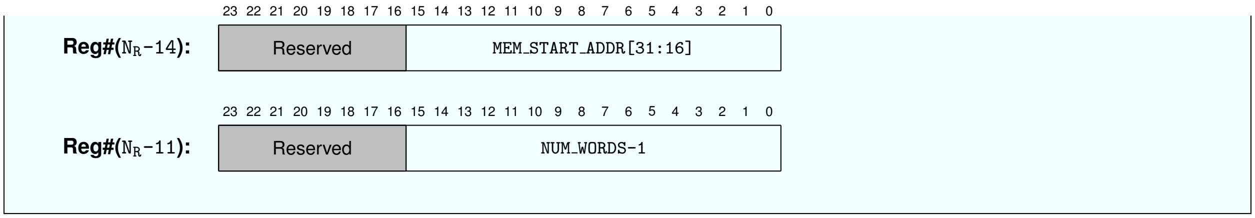
For example, if N_R=60 and the layer is receiving a Streaming Write message with STR_CH=1, then the first MEM_DATA (i.e., the first 32-bit of MBUS_DATA) is written to

{Write Buffer[31:16](Reg#38), Write Buffer[15:2](Reg#37), 2'h0} + ({Buffer Offset[15:0](Reg#40)} << 2)

Once it is written, Buffer Offset value is incremented by 1 by Layer Ctrl. Thus, the next MEM_DATA is written to the subsequent memory address.

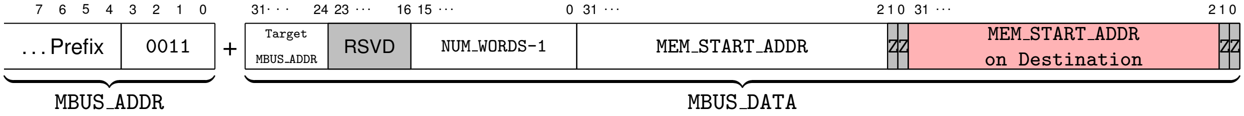
Status Registers A part of MBus Support Registers is used for status registers. When the write completes, Layer Ctrl updates the status registers as shown below.

<table>
<tr><td></td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td><b>Reg#(N_R-15):</b></td><td colspan="4">4 MSB of Prefix</td><td colspan="2">01</td><td colspan="2">STR CH</td><td colspan="14">MEM_START_ADDR[15:2]</td><td>RSV</td><td>RSV</td></tr>
</table>



Overflow If more data is received after writing the last address in the 32-bit address space (0xFFFFFFFC), the destination address wraps and data continues to be written, beginning at address 0x00000000.

7.6.3 Memory Read Message



- Target MBUS_ADDR is the MBUS_ADDR to be used in the resulting response message.
- NUM_WORDS-1 is the length of the requested read in words less one. A length of 0 will reply with 1 words of data.
- MEM_START_ADDR is the address in memory to read from. The bottom two bits of the address field are reserved and **must** be transmitted as 0.
- MEM_START_ADDR on Destination is **optional**. If it is present, it is prepended to the reply (generating a message with a Memory Bulk Write Message formatted payload). If it is omitted, data is immediately placed on the bus (generating a message with a Memory Stream Write Message formatted payload).

Figure 20 shows an example of a memory read transaction and shows how the response message (Memory Bulk Write) is made.

Figure 21 shows an example of a memory read transaction and shows how the response message (Memory Stream Write) is made.

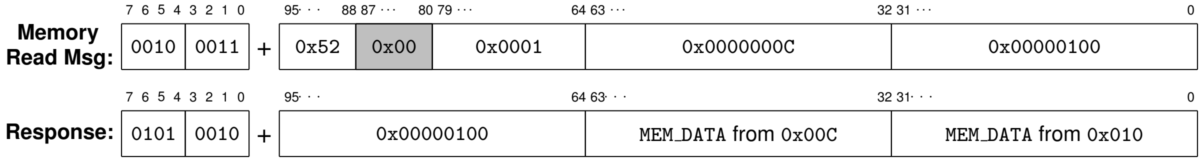


Figure 20: Example of MBus Memory Read message followed by Memory Bulk Write message
Target MBUS_ADDR (MBUS_DATA[95:88]) in Memory Read Msg is directly copied into MBUS_ADDR section in the Response. MEM_START_ADDR on Destination (MBUS_DATA[31:0]) in Memory Read Msg is directly copied into MBUS_DATA[95:64] in the Response. NUM_WORDS-1 (MBUS_DATA[79:64]) in Memory Read Msg is set to 1, so it reads 2 words starting from the MEM_START_ADDR (MBUS_DATA[63:32]) specified in Memory Read Msg, resulting in the Response which contains the data from the memory address 0x00C and 0x100. Note that the Response is in the form of Memory Bulk Write Message.

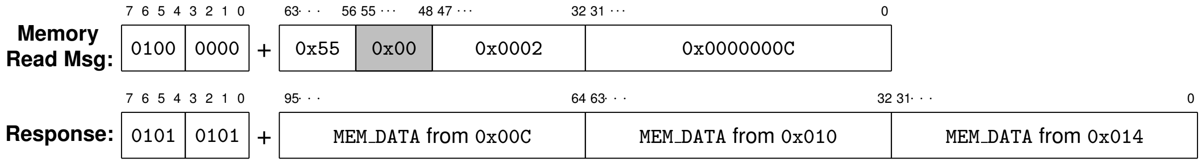
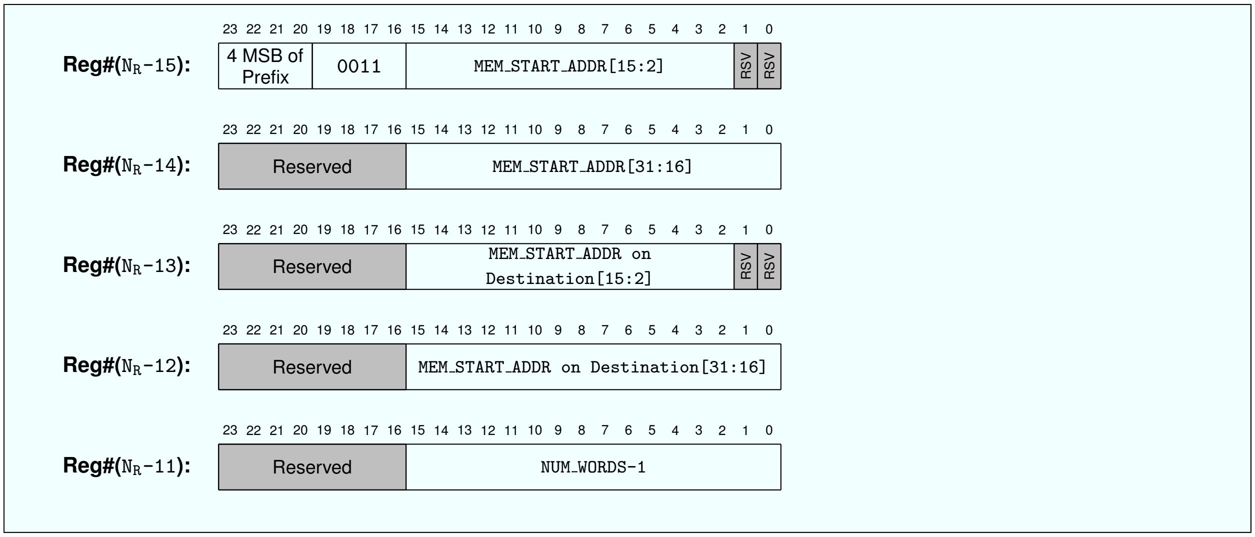


Figure 21: Example of MBus Memory Read message followed by Memory Stream Write message
Note that the MEM_START_ADDR on Destination field is missing in the Memory Read Msg, thus, the Response only contains memory data, without any memory address information.
Target MBUS_ADDR (MBUS_DATA[63:56]) in Memory Read Msg is directly copied into MBUS_ADDR section in the Response. NUM_WORDS-1 (MBUS_DATA[79:64]) in Memory Read Msg is set to 2, so it reads 3 words starting from the MEM_START_ADDR (MBUS_DATA[31:0]) specified in Memory Read Msg, resulting in the Response which contains the data from the memory address 0x00C, 0x100, and 0x104. Note that the Response is in the form of Memory Stream Write Message.

Status Registers A part of MBus Support Registers is used for status registers. When the response completes, Layer Ctrl updates the following MBus Support Registers.



Overflow If MEM_START_ADDR field and subsequent length exceed the 32-bit address space (0x100000000), the address wraps and data continues to be read from, beginning at address 0x00000000.

7.7 Reserved MBus Support Registers

The original MBus specification defines a few more MBus Support Registers. However, in order to achieve power/logic reduction, not all of them have been actually implemented.

This section describes the definition of the MBus Support Registers that are missing from the current MBus implementation. Although they are referred to 'Reserved', it is fine to use these REG_ID s for other purposes, since the current MBus Bus Ctrl and Layer Ctrl implementation does not attempt to access or use these REG_ID s for their own purposes.

7.7.1 Record and Interrupt Control

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reg#(N_R-35):	Reserved																INT	INT	INT	INT	INT	INT	INT	INT

Each time a layer completes a Register/Memory command, it checks the associated command `INT` bit. If `1'b1`, the layer is interrupted.

7.7.2 Action Register

	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reg#(N_R-0):	RST	RSV			RSTR	RSTB	RSTS	Reserved								INTO	Reserved							

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 behavior of anything after a register action request in the same transaction is undefined.

- `Reset (RST)`
  - Reset the entire layer. The exact result of a reset is implementation-defined, however this request should be the most aggressive form of reset available.
- `Reset MBus Support Registers (RSTR)`
  - Reset MBus Support Registers to their default value.
  - Resets Reg#(N_R-32) ∼ Reg#(N_R-8).
- `Reset MBus Support Registers (Memory Bulk) (RSTB)`
  - Reset MBus Support Registers that affect memory bulk transfers to their default value.
  - Resets Reg#(N_R-13).
- `Reset MBus Support Registers (Memory Stream) (RSTS)`
  - Reset MBus Support Registers that affect memory stream transfers to their default value.
  - Resets Reg#(N_R-31) ∼ Reg#(N_R-16).
- `Interrupt Owner (INTO)`
  - Asserts the Layer Interrupt.

8 Layer Controller

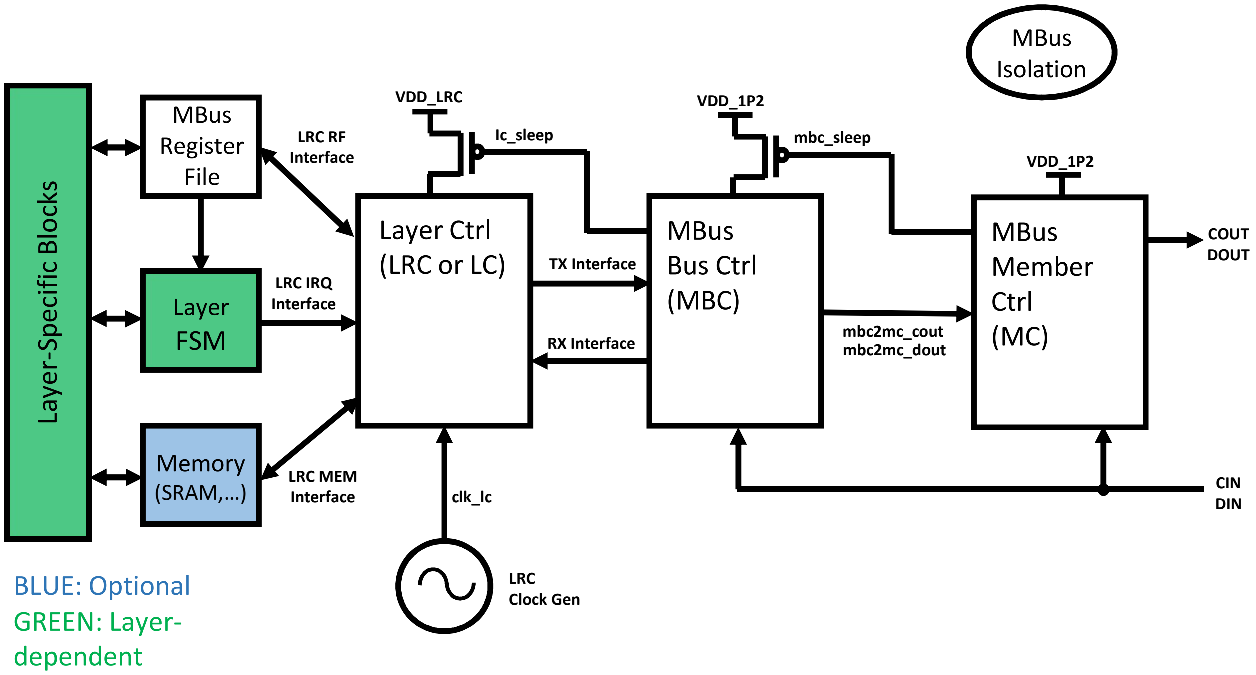


Figure 22: Member layer block diagram

As shown in Figure 22, Layer Ctrl has 5 types of interfaces:

- TX Interface
- RX Interface
- Interrupt Interface
- Register File (RF) Interface
- Memory (MEM) Interface

This document explains only about the Interrupt Interface. Details of other interfaces are included in a separate document[40].

8.1 Interrupt Interface

The Interrupt Interface of Layer Ctrl implements the typical asynchronous handshake interface between Layer Ctrl and Layer FSM. Figure 23 shows the Interrupt Interface between Layer Ctrl and Layer FSM.

It is designed to be used when the Layer FSM needs to do the following:

- Write into MBus Register File
- Read from MBus Register File
- Read from Memory

[40] *MBus Implementation Guide* (pending).

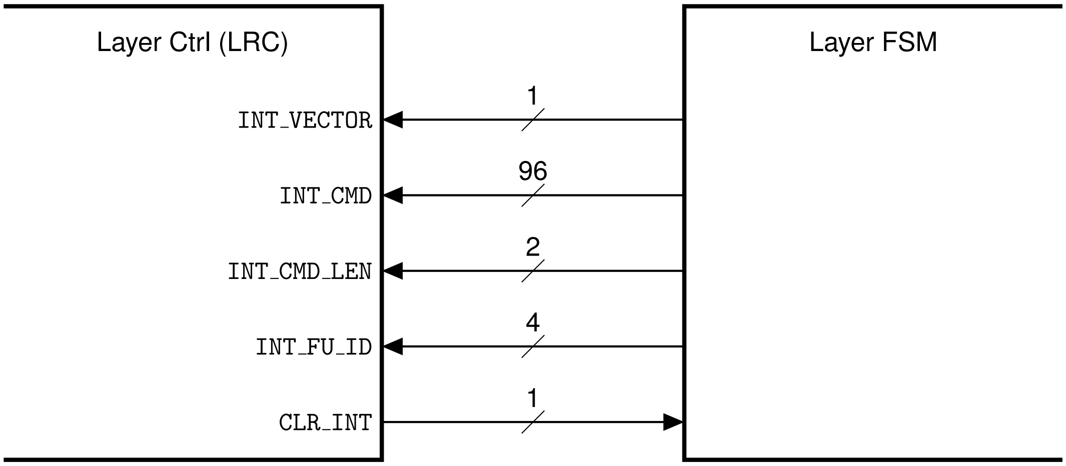


Figure 23: Layer Controller Interrupt Interface

Note that it does *not* support the Memory Write operation. The Layer FSM, if needed, has to have its own interface with the Memory. Some arbitration between the Layer FSM and Layer Ctrl may be required.

Layer Ctrl and Layer FSM do not need to use the same clock. If they use separate clocks, then they are regarded to be asynchronous, and proper synchronization scheme must be implemented while handling the Interrupt Interface. If they use the same clock, then the synchronization scheme may not be needed.

Layer Ctrl and Layer FSM do not need to be in the same power domain. If they are in different power domains, then proper isolation or level conversion scheme must be implemented while handling the Interrupt Interface. If they are in the same power domain, then the signal isolation and level conversion may not be required.

The Interrupt Interface consists of 5 signals:

- From Layer FSM to Layer Ctrl
  - `INT_VECTOR` (1 bit): Interrupt Request signal
  - `INT_CMD` (96 bits): Interrupt Command
  - `INT_CMD_LEN` (2 bits): Interrupt Command Length
  - `INT_FU_ID` (4 bits): Interrupt Functional Unit ID (`FU_ID`)
- From Layer Ctrl to Layer FSM
  - `CLR_INT` (1 bit): Clear Interrupt

`INT_VECTOR` is an interrupt request signal, and `CLR_INT` is an clear interrupt signal. Whenever Layer FSM needs to interrupt Layer Ctrl, it needs to first prepare the Interrupt Signals (i.e., `INT_CMD`, `INT_CMD_LEN`, `INT_FU_ID`), and then set `INT_VECTOR =1'b1`. This will trigger Layer Ctrl to go into the Interrupt Handling mode. Once Layer Ctrl has done with the interrupt, Layer Ctrl raises `CLR_INT`. It is Layer FSM's responsibility to hold `INT_VECTOR =1'b1` *until* Layer Ctrl sets `CLR_INT =1'b1`. Once Layer FSM detects `CLR_INT =1'b1`, it *must* resets `INT_VECTOR =1'b0`, so that Layer Ctrl also resets `CLR_INT =1'b0`.

`INT_CMD`, `INT_CMD_LEN`, and `INT_FU_ID` signals must be stable when Layer Ctrl detects `INT_VECTOR =1'b1`.

Once Layer Ctrl detects `INT_VECTOR =1'b1`, it behaves *as if* it had received an MBus message. Figure 24 shows the equivalent MBus messages corresponding to the Interrupt Interrupt signal set.

'Short Prefix' shown in the figure represents this layer's short prefix. Actually, the Interrupt Interface does not have to know the currently assigned short prefix since it bypasses MBus Bus Ctrl and directly interfaces with

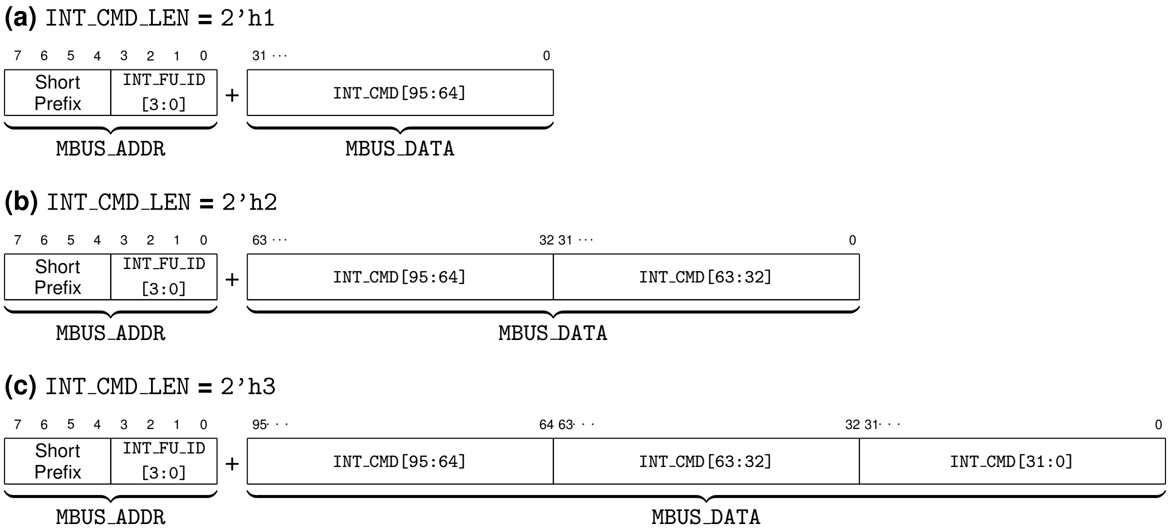


Figure 24: Interrupt Interface Signal Set and Equivalent MBus Messages

Layer Ctrl.[41]

Following sections describe how to set the Interrupt Interface signal set in various scenarios.

8.1.1 Write into MBus Register File

This is used when Layer FSM needs to write in up to 3 registers in MBus Register File. Table 5 shows how to set each Interrupt signals.

Table 5: Interrupt Signals for Write into MBus Register File

Signal		Write into 1 Register	Write into 2 Registers	Write into 3 Registers
INT_FU_ID		4'h0	4'h0	4'h0
INT_CMD_LEN		2'h1	2'h2	2'h3
INT_CMD	[95:88]	REG_ID_0	REG_ID_0	REG_ID_0
	[87:64]	REG_DATA_0	REG_DATA_0	REG_DATA_0
	[63:56]	Ignored	REG_ID_1	REG_ID_1
	[55:32]	Ignored	REG_DATA_1	REG_DATA_1
	[31:24]	Ignored	Ignored	REG_ID_2
	[23:0]	Ignored	Ignored	REG_DATA_2

* REG_DATA_N is written to the register whose ID is REG_ID_N (N=0–2)

Note that Table 5 contains the exactly same information shown in Register Write Message bitfield representation, except the short prefix.

Once these signals are set, Layer FSM needs to raise INT_VECTOR. Section 8.1.7 describes the detailed timing waveforms for the interrupt signals.

Once Layer Ctrl takes this interrupt, it will write into MBus Register File as specified in INT_CMD. Once the write operation is done, Layer Ctrl raises CLR_INT. There is no MBus message resulting from this interrupt.

[41] It is MBus Bus Ctrl's role to check incoming MBus message's MBUS_ADDR. If the MBUS_ADDR contains the same short prefix as the layer's, MBus Bus Ctrl transfers the MBus message to Layer Ctrl.

8.1.2 Read from MBus Register File

This is used when Layer FSM needs to read from MBus Register File *and* send the read data to another layer using an MBus message. Table 6 shows how to set each Interrupt signals.

Table 6: Interrupt Signals for Read from MBus Register File

Signal		Description
INT_FU_ID		4'h1
INT_CMD_LEN		2'h1
INT_CMD	[95:88]	First REG_ID to Read From
	[87:80]	NUM_REGS-1
	[79:72]	Target MBUS_ADDR
	[71:64]	Target REG_ID on Destination
	[63:0]	Ignored

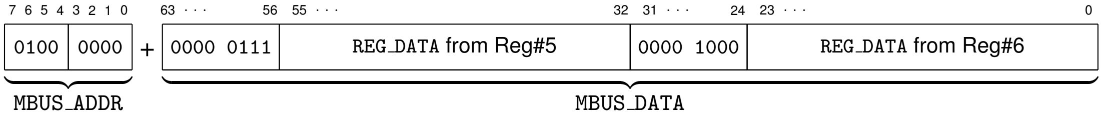
Note that Table 6 contains the exactly same information shown in Register Read Message bitfield representation, except the short prefix.

Once these signals are set, Layer FSM needs to raise INT_VECTOR. Section 8.1.7 describes the detailed timing waveforms for the interrupt signals.

Once Layer Ctrl takes this interrupt, it reads NUM_REGS registers in MBus Register File, starting from First REG_ID to Read From. Then, Layer Ctrl generates an MBus message, in the form of Register Write Message, whose MBUS_ADDR is Target MBUS_ADDR and MBUS_DATA composed of the Target REG_ID on Destination and the data read from MBus Register File. Figure 25 shows an example.

Signal		Value	Description
INT_CMD	[95:88]	8'h05	First REG_ID to Read From
	[87:80]	8'h01	NUM_REGS-1
	[79:72]	8'h40	Target MBUS_ADDR
	[71:64]	8'h07	Target REG_ID on Destination
	[63:0]	64'h0	Ignored

(a) Interrupt Signals



(b) Resulting MBus Message

Figure 25: Example of MBus Register Read Interrupt INT_CMD[71:64] becomes MBUS_ADDR in the resulting MBus message. Layer Ctrl reads NUM_REGS registers, which is INT_CMD[87:80] plus 1, starting from the REG_ID specified in INT_CMD[95:88]. The resulting MBus message is in the form of Register Write Message. MBUS_DATA[63:56] is copied from INT_CMD[71:64], and MBUS_DATA[55:32] is the data read from the register specified in INT_CMD[95:88] (i.e., Reg#5 in this example). MBUS_DATA[31:24] is MBUS_DATA[63:56] incremented by 1, and MBUS_DATA[23:0] is the data read from the subsequent register (i.e., Reg#6 in this example).

8.1.3 Read from Memory

This is used when Layer FSM needs to read from Memory *and* send the read data to another layer using an MBus message.

Note that Layer FSM may have a separate memory read interface, probably with some kind of arbitration (since Layer Ctrl also has the memory read interface — LRC MEM Interface). In most cases, it is more efficient and faster for Layer FSM to use this dedicated memory read interface if the sole purpose is to *just* read from the memory.

One of the purposes of Layer Ctrl providing the Memory Read Interrupt, is to let Layer FSM be able to send an arbitrary MBus message to other layers. As a result of this Memory Read Interrupt, Layer Ctrl generates an MBus message. By storing appropriate data in the proper memory location, Layer FSM can decide which data should be contained in the MBUS_DATA section of the resulting MBus message. This will be further detailed in Section 8.1.6.

The resulting MBus message is in the form of either Memory Bulk Write Message or Memory Stream Write Message, which will be described in the following sections.

8.1.4 Read from Memory: To Generate Memory Bulk Write Message

Table 7 shows how to set each Interrupt signals.

Table 7: Interrupt Signals to Generate Memory Bulk Write Message

Signal		Description
INT_FU_ID		4'h3
INT_CMD_LEN		2'h3
INT_CMD	[95:88]	Target MBUS_ADDR
	[87:80]	8'h0
	[79:64]	NUM_WORDS-1
	[63:32]	MEM_START_ADDR
	[31:0]	MEM_START_ADDR on Destination

Note that Table 7 contains the exactly same information shown in Memory Read Message bitfield representation, except the short prefix.

Once these signals are set, Layer FSM needs to raise INT_VECTOR. Section 8.1.7 describes the detailed timing waveforms for the interrupt signals.

Once Layer Ctrl takes this interrupt, it reads NUM_WORDS words from the memory, starting from the address given in MEM_START_ADDR (INT_CMD[63:32]). Then, Layer Ctrl generates an MBus message, in the form of Memory Bulk Write Message, whose MBUS_ADDR is Target MBUS_ADDR (INT_CMD[95:88]), and MBUS_DATA composed of the MEM_START_ADDR on Destination (INT_CMD[31:0]) followed by the memory read data. Figure 26 shows an example.

8.1.5 Read from Memory: To Generate Memory Stream Write Message

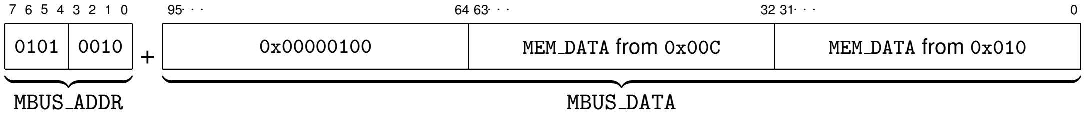
Table 8 shows how to set each Interrupt signals.

Table 8: Interrupt Signals to Generate Memory Stream Write Message

Signal		Description
INT_FU_ID		4'h3
INT_CMD_LEN		2'h2
INT_CMD	[95:88]	Target MBUS_ADDR
	[87:80]	8'h0
	[79:64]	NUM_WORDS-1
	[63:32]	MEM_START_ADDR
	[31:0]	Ignored

Signal		Value	Description
INT_CMD	[95:88]	8’h52	Target MBUS_ADDR
	[87:80]	8’h0	Reserved
	[79:64]	16’h1	NUM_WORDS-1
	[63:32]	32’h00C	MEM_START_ADDR
	[31:0]	32’h100	MEM_START_ADDR on Destination

(a) Interrupt Signals



(b) Resulting MBus Message

Figure 26: Example of MBus Memory Read Interrupt Resulting in Memory Bulk Write Message INT_CMD[95:88] becomes MBUS_ADDR in the resulting MBus message. Layer Ctrl reads NUM_WORDS word, which is INT_CMD[79:64] plus 1, starting from MEM_START_ADDR specified in INT_CMD[63:32]. The resulting MBus message is in the form of Memory Bulk Write Message. MBUS_DATA[95:64] is copied from INT_CMD[31:0]. MBUS_DATA[63:32] is the memory data read from MEM_START_ADDR (INT_CMD[63:32]), and MBUS_DATA[31:0] is the memory data read from MEM_START_ADDR+4. If NUM_WORDS-1 in INT_CMD[79:64] was 16’h2, then the resulting MBus message would have had another subsequent MBUS_DATA section, which is the memory data read from MEM_START_ADDR+8.

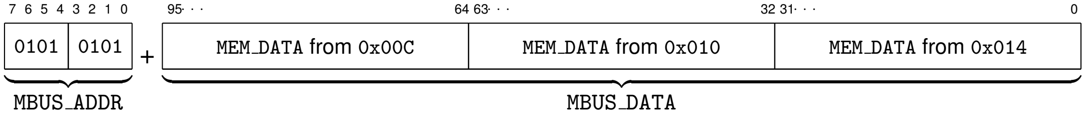
Note that Table 8 contains the exactly same information shown in Memory Read Message bitfield representation, except the short prefix.

Once these signals are set, Layer FSM needs to raise INT_VECTOR. Section 8.1.7 describes the detailed timing waveforms for the interrupt signals.

Once Layer Ctrl takes this interrupt, it reads NUM_WORDS words from the memory, starting from the address given in MEM_START_ADDR (INT_CMD[63:32]) field. Then, Layer Ctrl generates an MBus message, in the form of Memory Stream Write Message, whose MBUS_ADDR is Target MBUS_ADDR (INT_CMD[95:88]), and MBUS_DATA composed of the memory read data. Figure 27 shows an example.

Signal		Value	Description
INT_CMD	[95:88]	8’h55	Target MBUS_ADDR
	[87:80]	8’h0	Reserved
	[79:64]	16’h2	NUM_WORDS-1
	[63:32]	32’hC	MEM_START_ADDR
	[31:0]	32’h0	Ignored

(a) Interrupt Signals



(b) Resulting MBus Message

Figure 27: Example of MBus Memory Read Interrupt Resulting in Memory Stream Write Message INT_CMD[95:88] becomes MBUS_ADDR in the resulting MBus message. Layer Ctrl reads NUM_WORDS word, which is INT_CMD[79:64] plus 1, starting from MEM_START_ADDR specified in INT_CMD[63:32]. The resulting MBus message is in the form of Memory Stream Write Message. MBUS_DATA[95:64] is the memory data read from MEM_START_ADDR (INT_CMD[63:32]), MBUS_DATA[63:32] is the memory data read from MEM_START_ADDR+4, and MBUS_DATA[31:0] is the memory data read from MEM_START_ADDR+8, If NUM_WORDS-1 in INT_CMD[79:64] was 16’h3, then the resulting MBus message would have had another subsequent MBUS_DATA section, which is the memory data read from MEM_START_ADDR+12.

8.1.6 Read from Memory: To Generate Arbitrary Message

The Memory Read Interrupt can be used to generate arbitrary MBus messages. This utilizes the Memory Read Interrupt that generates an MBus Memory Stream Write Message (Section 8.1.5), but with arbitrary `Target MBUS_ADDR`. The Layer Ctrl does not check the validity of the `FU_ID` contained in the `Target MBUS_ADDR`, so it can be just an arbitrary value. In addition, the resulting Stream Write message does not contain a target memory address; it just contains the read data from the memory, which is useful when trying to send an arbitrary MBus message.

Below is the steps to generate arbitrary MBus messages.

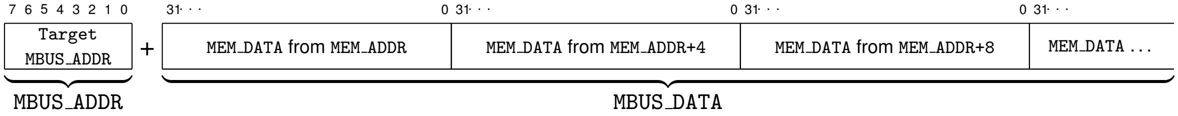
1. Save the data to be sent in a memory. This data will become `MBUS_DATA` section of the resulting MBus message. There is no restriction on the memory address (`MEM_ADDR`) where this data is stored, as long as Layer Ctrl can access the `MEM_ADDR` *and* the `MEM_ADDR` is known to Layer FSM.
2. Set `INT_FU_ID`, `INT_CMD_LEN`, and `INT_CMD` appropriately. See Table 9.
3. Set `INT_VECTOR =1'b1` and wait until `CLR_INT` becomes `1'b1`.
4. Reset `INT_VECTOR =1'b0` and wait until `CLR_INT` becomes `1'b0`.

Table 9: Interrupt Signals to Generate Arbitrary Message

Signal		Description
`INT_FU_ID`		`4'h3`
`INT_CMD_LEN`		`2'h2`
`INT_CMD`	`[95:88]`	`Target MBUS_ADDR`
	`[87:80]`	`8'h0`
	`[79:64]`	`NUM_WORDS-1`
	`[63:32]`	`MEM_ADDR`
	`[31:0]`	Ignored

`Target MBUS_ADDR` becomes the `MBUS_ADDR` of the resulting MBus message. `NUM_WORDS-1` (`INT_CMD[79:64]`) should be the 'number of 32-bit chunks in `MBUS_DATA` minus 1'. `MEM_ADDR` is the memory address where the data to be sent is stored.

The resulting MBus message is shown below.



Note that user has the full control on both `MBUS_ADDR` and `MBUS_DATA` of the resulting MBus message. The length of the `MBUS_DATA` section can be also controlled using `NUM_WORDS-1` (`INT_CMD[79:64]`)'.

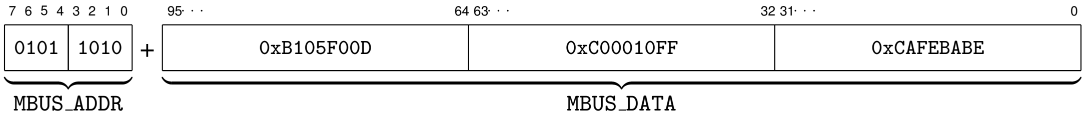
Figure 28 and Figure 29 show a couple examples.

MEM_ADDR	MEM_DATA
0x00000904	0xB105F00D
0x00000908	0xC00010FF
0x0000090C	0xCAFEBABE

(a) Memory Contents

Signal		Value	Description
INT_CMD	[95:88]	8'h5A	Target MBUS_ADDR
	[87:80]	8'h00	Reserved
	[79:64]	16'h0002	NUM_WORDS-1
	[63:32]	32'h00000904	MEM_ADDR
	[31:0]	32'h00000000	Ignored

(b) Interrupt Signals



(c) Resulting MBus Message

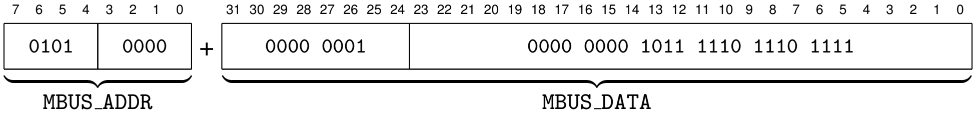
Figure 28: Example of MBus Memory Read Interrupt Resulting in an Arbitrary MBus Message INT_CMD[95:88] becomes MBUS_ADDR in the resulting MBus message. Note that Layer Ctrl does not care about the validity of FU_ID used in this MBUS_ADDR. In this example, FU_ID =4'hA, which is invalid, and other layers may just ignore this resulting MBus message. This just shows that you can generate **any** MBus messages using this Interrupt Interface. Since NUM_WORDS-1 is 2, Layer Ctrl reads 3 words starting from MEM_ADDR, and puts them in MBUS_DATA section of the resulting MBus message.

MEM_ADDR	MEM_DATA
0x00000400	0x0100BEEF

(a) Memory Contents

Signal		Value	Description
INT_CMD	[95:88]	8'h50	Target MBUS_ADDR
	[87:80]	8'h00	Reserved
	[79:64]	16'h0000	NUM_WORDS-1
	[63:32]	32'h00000400	MEM_ADDR
	[31:0]	32'h00000000	Ignored

(b) Interrupt Signals



(c) Resulting MBus Message

Figure 29: Example of MBus Memory Read Interrupt Resulting in a Register Write Message The resulting FU_ID (MBUS_ADDR[3:0] in the resulting MBus message) is set to 4'h0, because INT_CMD[91:88] is 4'h0. This FU_ID indicates 'Register Write' (Section 7.5.2). The MBUS_DATA section is directly copied from the memory at MEM_ADDR. Thus, the resulting MBus message is targeted for a layer whose short prefix is 4'h5 (MBUS_ADDR[7:4]), and it would write the data, 0x00BEEF (MBUS_DATA[23:0]), into the Reg#1 (MBUS_DATA[31:24]) in that layer.

8.1.7 INT_VECTOR and CLR_INT

Once the Interrupt Signals (i.e., INT_CMD, INT_CMD_LEN, INT_FU_ID) are set, Layer FSM must set INT_VECTOR =1'b1 to interrupt Layer Ctrl so that Layer Ctrl reads from these Interrupt Signals and does what is specified in these signals.

INT_VECTOR and CLR_INT timings assumes that Layer FSM and Layer Ctrl use different clocks, hence asynchronous to each other. When they are asynchronous, there must be a synchronization scheme included in this Interrupt Interface. If they are synchronous (i.e., they use the same clock), this synchronization scheme may be skipped, and the interrupt latency may be shortened.

Figure 30 shows typical waveforms used in the Interrupt Interface.

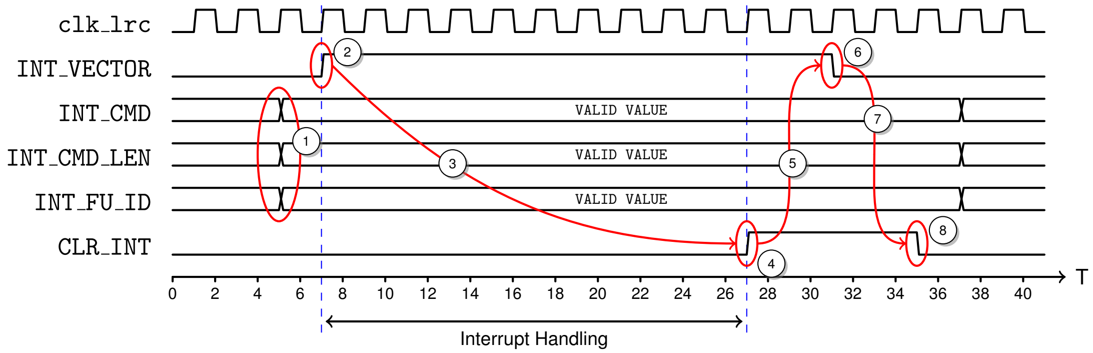


Figure 30: INT_VECTOR **and** CLR_INT **Timing** clk_lrc is the clock for Layer Ctrl. Layer FSM may use the same clock. If Layer FSM uses a different clock, then a proper synchronization scheme must be included between Layer Ctrl and Layer FSM.

Below is a step-by-step explanation on the numbered marks in the figure.

1. Layer FSM prepares the Interrupt Signals: INT_CMD, INT_CMD_LEN, and INT_FU_ID.
2. Layer FSM sets INT_VECTOR =1'b1. [42]
3. Layer Ctrl detects the interrupt and performs what is specified in the Interrupt Signals. It is noted 'Interrupt Handling' in the figure. If Layer FSM and Layer Ctrl are asynchronous, this also includes the synchronization delay.
4. Once Layer Ctrl finishes the interrupt handling, Layer Ctrl sets CLR_INT =1'b1.
5. It would take at least one clock cycle until Layer FSM detects CLR_INT =1'b1. If Layer FSM and Layer Ctrl are asynchronous, this delay may be longer due to the synchronization delay.
6. Once Layer FSM detects CLR_INT =1'b1, Layer FSM should reset INT_VECTOR =1'b0.
7. It would take at least one clock cycle until Layer Ctrl detects INT_VECTOR =1'b0. If Layer FSM and Layer Ctrl are asynchronous, this delay may be longer due to the synchronization delay.
8. Once Layer Ctrl detects CLR_INT =1'b0, Layer Ctrl resets CLR_INT =1'b0. Layer FSM may reset the Interrupt Signals as soon as it detects CLR_INT =1'b1. In this figure, Layer FSM resets the Interrupt Signals after it detects CLR_INT =1'b0.

[42] It should be noted that INT_VECTOR may be set 1'b1 at the same time the Interrupt Signals are set. If Layer FSM and Layer Ctrl are asynchronous, then there would be a synchronization delay until Layer Ctrl detects INT_VECTOR =1'b1, and the Interrupt Signal would become stable during this synchronization delay. Even if Layer FSM and Layer Ctrl are synchronous, it takes at least one clock cycle before Layer Ctrl detects INT_VECTOR =1'b1, and the one clock cycle delay should be enough for the Interrupt Signals to become stable. Hence, in practice, it should be fine to set INT_CMD, INT_CMD_LEN, INT_FU_ID, and INT_VECTOR altogether at the same time.

If the interrupt causes the Layer Ctrl to generate an MBus message, then Layer Ctrl resets `CLR_INT` *after* it completes sending the MBus message. Figure 31 shows an example of the timing waveform when the interrupt makes Layer Ctrl to send a Register Write message. Note that Layer Ctrl resets `CLR_INT` to `1'b0` *after* the message transmission is completed.

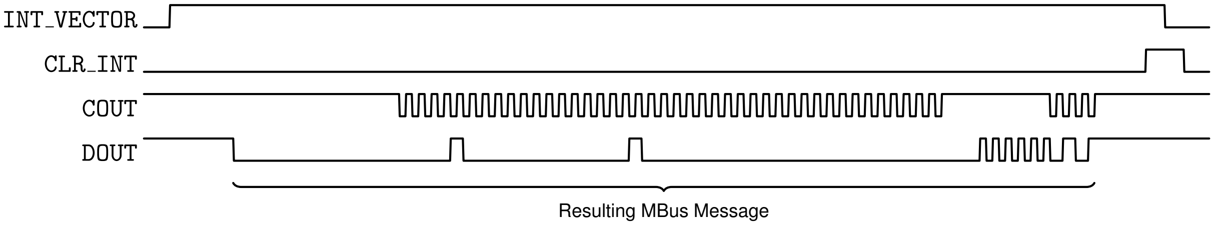


Figure 31: `INT_VECTOR` **and** `CLR_INT` **Timing with a Resulting MBus Message** Layer Ctrl resets `CLR_INT` *after* it finishes sending the MBus message. The resulting MBus message in this example reads `MBUS_ADDR =0x40`, `MBUS_DATA =0x01000000`, which is a Register Write Message, and would write `0x000000` in Register#1 in a layer whose short prefix is `0x4`.

8.1.8 Multiple Interrupts

MBus supports multiple sets of Interrupt Signals. In this case, `INT_VECTOR` and `CLR_INT` are also multi-bit signals, and LSB of `INT_VECTOR` (i.e., `INT_VECTOR[0]`) has the highest priority.

If a layer has N sets of Interrupt Signals, each Interrupt Signals shall be extended as below:

- INT_CMD[96×N-1:0]
- INT_CMD_LEN[2×N-1:0]
- INT_FU_ID[4×N-1:0]
- INT_VECTOR[N-1:0]
- CLR_INT[N-1:0]

Then, INT_VECTOR[n] (n=0, 1, ..., $N-1$) is accompanied with:

- INT_CMD[96×$(n+1)$-1:0]
- INT_CMD_LEN[2×$(n+1)$-1:0]
- INT_FU_ID[4×$(n+1)$-1:0]
- CLR_INT[n]

Each sets are completely independent to each other. Within each set, the signal timings are same as described in previous sections.

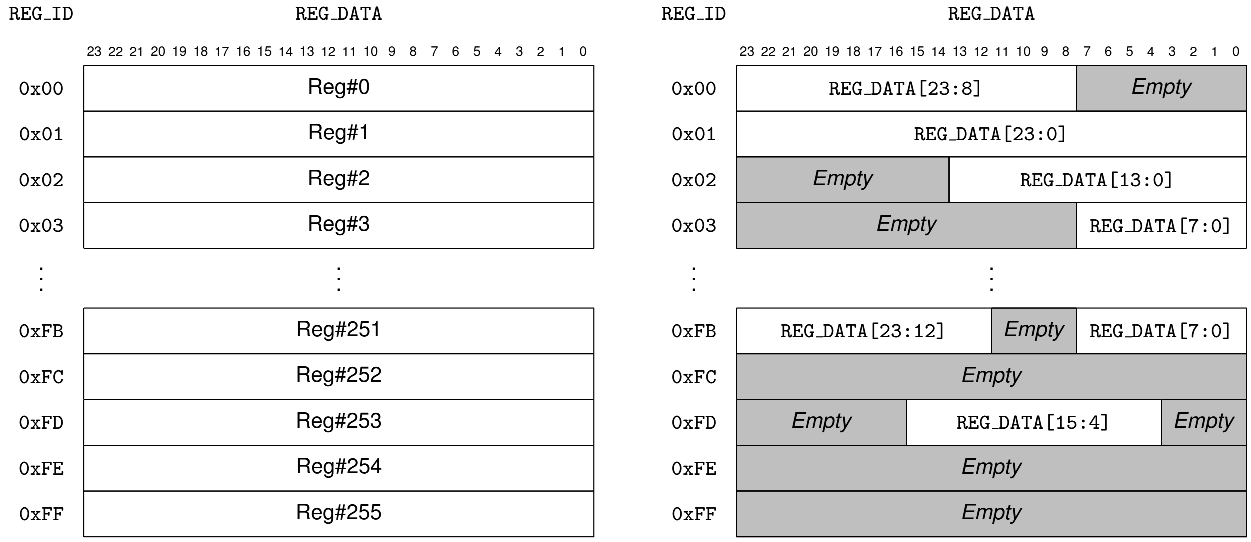
9 MBus Register File

As mentioned in Section 7.5.1, MBus Register File is one of the main components in MBus implementation as shown in Figure 3 and Figure 4.

9.1 Specification

MBus Register File is a group of registers:

- There can be up to 256 registers.
- Each register has a unique 8-bit `REG_ID` (0 ∼ 255). Most M^3 layers have far less than 256 registers, and the `REG_ID` of the last register (i.e., the register with the largest `REG_ID`) is referred to $\mathtt{N_R}$.
- Each register can have up to 24 bits. Thus, `REG_DATA` is 24-bit-wide.
- It is allowed to have one or more empty (un-used) registers.
- It is allowed to have one or more empty (un-used) bits within a register.
- Un-used bits or registers are treated as RZWI (Read as Zero, Write Ignored).



REG_ID	REG_DATA (23 … 0)
0x00	Reg#0
0x01	Reg#1
0x02	Reg#2
0x03	Reg#3
⋮	⋮
0xFB	Reg#251
0xFC	Reg#252
0xFD	Reg#253
0xFE	Reg#254
0xFF	Reg#255

(a) MBus Register File Structure

REG_ID	REG_DATA (23 … 0)
0x00	REG_DATA[23:8] *Empty*
0x01	REG_DATA[23:0]
0x02	*Empty* REG_DATA[13:0]
0x03	*Empty* REG_DATA[7:0]
⋮	⋮
0xFB	REG_DATA[23:12] *Empty* REG_DATA[7:0]
0xFC	*Empty*
0xFD	*Empty* REG_DATA[15:4] *Empty*
0xFE	*Empty*
0xFF	*Empty*

(b) MBus Register File Example ($\mathtt{N_R}$ =`0xFD` (253))

Figure 32: MBus Register File Structure and Example The example shows that it is possible to have empty registers and bits in the middle. $\mathtt{N_R}$ indicates the last `REG_ID` that is not completely empty, and in this example, $\mathtt{N_R}$ is 253. Empty bits and registers are treated as RZWI (Read as Zero, Write Ignored).

9.2 Types of MBus Register File

In the current MBus implementation, MBus Register File can have the types shown in Table 10. Following sections describe further details.

Table 10: Types of MBus Register File

Type	Write/Read	Power Domain	Retentive-ness	Isolation
W	Write/Read	PD_1P2	Retentive	N/A
R	Read-Only	Design-Dependent	Design-Dependent	Design-Dependent
MM	Write-Only	PD_LRC	N/A	N/A
NR	Write/Read	PD_LRC	Non-Retentive	No
NRI	Write/Read	PD_LRC	Non-Retentive	Yes
Pn (n=2–4)	Write/Read	PD_RFn (n=2–4)	User-Controlled	Yes

9.2.1 W-Type

W registers are Write-able and Read-able.

Usually they are implemented using low-leakage flip-flops. They are in PD_1P2 power domain, which is always-on, thus retentive.

Upon power-on, they reset to their default values. User can change their values in Active mode, and they retain their values in Sleep mode.

They do not need isolation scheme as they are always-on; they always properly outputting their current values.

9.2.2 R-Type

R registers are Read-Only registers. Any write operation into these registers are ignored.

They do not have a physical flip-flop in MBus Register File module, and thus, MBus Register File module itself does not have output pins for R registers. However, its REG_ID is still assigned and valid, so that Layer Ctrl recognize them as MBus Register File registers, which means, there are physical wires going into the Layer Ctrl input pins.

Thus, there must be other custom blocks in the layer who drives these wires. It is completely design-dependent, so the blocks driving these wires can be in any power domain.

If these blocks are in a power domain other than the Layer Ctrl power domain (PD_LRC), a proper isolation scheme *must* be implemented.

This also implies that the retentiveness of R registers is determined by the characteristics of the driving blocks' power domain.

If the power domain becomes power-gated (i.e., loses the power) in certain operations, they are not considered 'retentive.'

9.2.3 MM-Type

MM registers are Write-Only registers. Any read operation returns zero.

They do not have a physical flip-flop in MBus Register File module, and the *entire* register is empty (i.e., all the 24-bits are empty). However, its REG_ID is still assigned and valid, so that Layer Ctrl recognize them as MBus Register File registers, which means, Layer Ctrl still has an output for 'load' signal for these registers.

For example, if Reg#3 is an MM register, and there is an MBus Register Write message that writes into Reg#3, then Layer Ctrl sets the load signal for Reg#3 to 1'b1, even though there is no physical flip-flop assigned for Reg#3 in MBus Register File module. The REG_DATA contained in MBUS_DATA section of the incoming MBus message is ignored.

It is completely design-depended what to do with this load signal. Layer FSM may use this signal to initiate certain operations.

The load signal is a two-clock-cycle-wide (based on clk_lrc) pulse driven by Layer Ctrl, so MM registers are considered to be in the Layer Ctrl power domain (PD_LRC) and non-retentive.

If a block using this load signal is in a power domain other than PD_LRC, a proper isolation scheme *must* be implemented.

9.2.4 NR-Type

NR registers are Write-able and Read-able, and they are the Layer Ctrl power domain (PD_LRC). Because of this, they become powered-off and lose their values whenever Layer Ctrl powers off (i.e., in Sleep mode). Usually they are implemented using standard flip-flops.

NR registers do **not** have an isolation, and their output becomes *floating* whenever PD_LRC loses power. Therefore, they are regarded as 'non-retentive' registers, and their output *must not* be used in blocks outside of PD_LRC power domain.

Upon wake-up, they are always reset to their default values.

9.2.5 NRI-Type

NRI registers are Write-able and Read-able, and they are the Layer Ctrl power domain (PD_LRC). Because of this, they become powered-off and lose their values whenever Layer Ctrl powers off (i.e., in Sleep mode). Usually they are implemented using standard flip-flops.

NRI registers have an isolation, so their output becomes their default value whenever PD_LRC loses power. Because they lose their values and go back to their default values when losing power, they are regarded as 'non-retentive' registers. However, they always hold full-rail output voltages, so there is no isolation issue when blocks outside of PD_LRC power domain use NRI registers' outputs.

Upon wake-up, they are always reset to their default values.

9.2.6 Pn-Type (n=2–4)

Pn registers are Write-able and Read-able, and they are in their own power domain (PD_RFn (n=2–4)). Thus, their retentiveness is completely design-dependent.

The current MBus implementation supports up to 3 power domains for Pn registers, as shown in Table 11.

Table 11: Pn (n=2–4) Registers and Their Power Domain

Type	Power Domain	Sleep Signal	Isolation Signal
P2-Type	PD_RF2	RF2_SLEEP	RF2_ISOLATE
P3-Type	PD_RF3	RF3_SLEEP	RF3_ISOLATE
P4-Type	PD_RF4	RF4_SLEEP	RF4_ISOLATE

The sleep signals (RFn_SLEEP (n=2–4)) and the isolation signals (RFn_ISOLATE (n=2–4)) must be always valid. Thus, usually, these signals are driven by W-type registers.

PD_RFn power domain becomes powered off when the corresponding sleep signal becomes 1'b1. It is designer's responsibility to implement proper isolation scheme for PD_RFn power domain using corresponding isolation signal.

Since the sleep signals (RFn_SLEEP) and the isolation signals (RFn_ISOLATE) are driven by W-type registers, it is user's responsibility to keep the correct power-up/down sequence, as shown below.

For each n=2–4:

- When Power-On PD_RFn
  1. Set RFn_SLEEP=1'b0.
  2. Set RFn_ISOLATE=1'b0.
- When Power-Off PD_RFn
  1. Set RFn_ISOLATE=1'b1.
  2. Set RFn_SLEEP=1'b1.

Table 12: Pn (n=2–4) Registers Output Values

RFn_SLEEP	RFn_ISOLATE	Pn Register Output
0	0	Previous Value
0	1	Default Value
1	0	Floating
1	1	Default Value

Table 12 shows their output value status depending on the sleep and isolate signal values. As shown in the table, RFn_ISOLATE must not be 1'b0 while RFn_SLEEP=1'b1, as it results in floating output values. Also, note that Pn registers lose their previous values once RFn_SLEEP becomes 1'b1.

If the proper power-up/down sequence is followed, they always hold full-rail output voltages, so there should not be any isolation issue when blocks outside of PD_RFn power domain use Pn registers' outputs.

9.2.7 Example

Figure 33 shows an example implementation of MBus Register File. This figure does not include Pn registers; they can be thought as W-type registers in their own power domains.

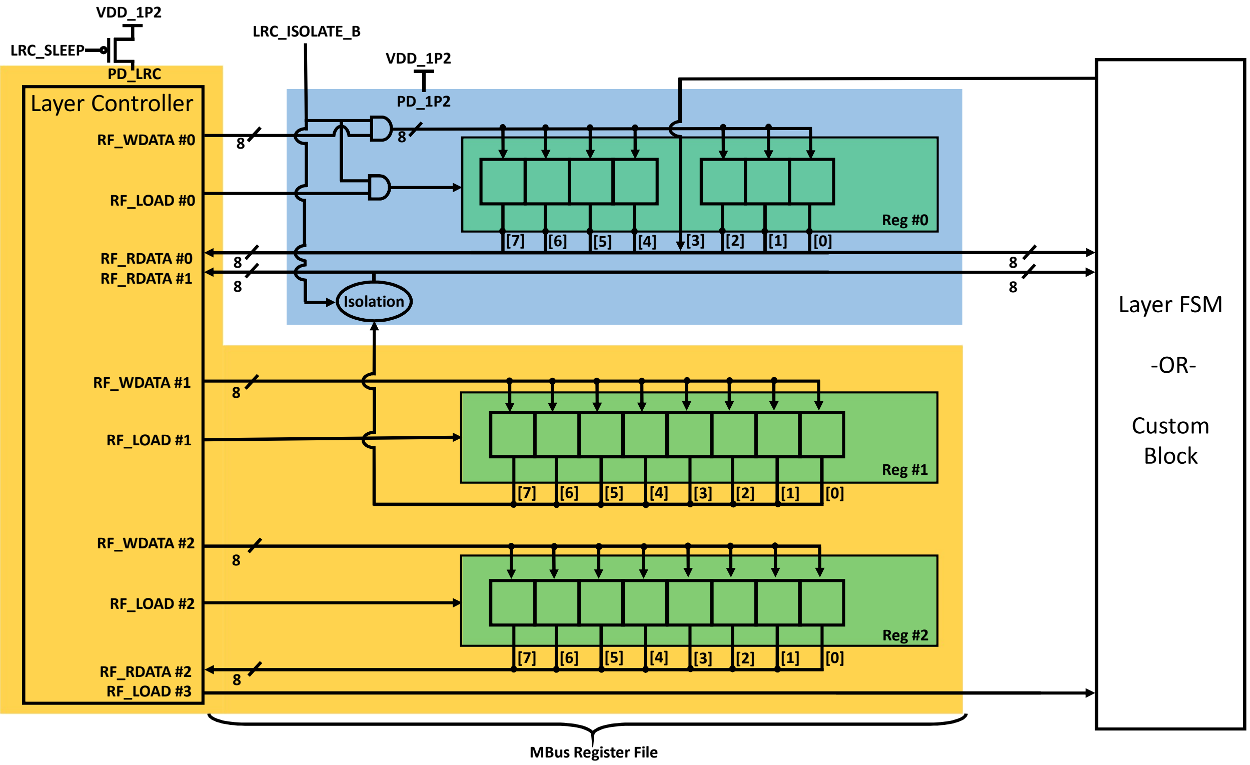


Figure 33: MBus Register File Example Reg#0 is an 8-bit register. Its [7:4] and [2:0] are `W`-type and in `PD_1P2` power domain, which is always-on, thus retentive. Its [3] is `R`-type, and it comes from Layer FSM or Custom Block, potentially from a different power domain. A proper isolation scheme must be implemented inside the Layer FSM or Custom Block, so that Layer Ctrl has a valid signal (`RF_RDATA#0`) whenever Layer Ctrl is powered on.
Reg#1 is an 8-bit `NRI`-type register. It is in the Layer Ctrl power domain (`PD_LRC`), but there is a dedicated isolation module inside `PD_1P2` power domain. Hence, its output (`RF_RDATA#1`) always has full-rail voltages, so that Layer FSM or Custom Block safely use this output without worrying about isolation issues. Note that it loses its previous value when `PD_LRC` turns off; it is isolated to its default value.
Reg#2 is an 8-bit `NR`-type register. It is mostly same as the `NRI`-type, but it does not have a dedicated isolation module. Therefore, its output becomes floating when `PD_LRC` turns off. It is **not** safe to use this output (`RF_RDATA#2`) in blocks that are not in `PD_LRC` power domain, unless a proper isolation scheme is implemented.
Reg#3 is an `MM`-type register. There is no physical flip-flop. However, Layer Ctrl still drives its load signal (`RF_LOAD#3`). It can be used, with a proper isolation implemented if necessary, in Layer FSM or other custom blocks.

Part II

Appendix

A C Files

B Layer Revision History

C M^3 Publications

D Document Revision History