An image containing logo

Auto-generated description

|  |
| --- |
|  |
| System Management Bus Core Specification |
|  |
|  |

**MASTERY**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Company** | **Name** | **Function** | **Date** |
| Written by: | ELSYS Design | Killian MICHAUD | SoC Design Engineer | 07/11/2024 |
| Approved by: |  |  |  |  |
| Authorized by: |  |  |  |  |

**REFERENCE DOCUMENT**

|  |  |
| --- | --- |
| **Title** | **Reference** |
| **System Management Bus (SMBus) Specification** | smbus20.pdf |
| **WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores** | wbspec\_b3.pdf |

Table of contents

[1. Introduction 5](#_Toc181966317)

[1.1. Presentation of the document 5](#_Toc181966318)

[1.2. Principle and application domain 5](#_Toc181966319)

[1.3. Targeted hardware 5](#_Toc181966320)

[2. Global description of module IP\_SMBus 6](#_Toc181966321)

[2.1. Architecture of module IP\_SMBus 6](#_Toc181966322)

[2.2. Functional diagram of module IP\_SMBus 6](#_Toc181966323)

[2.3. Specifications of module IP\_SMBus 7](#_Toc181966324)

[2.4. Module IP\_SMBus interface description (IO Ports) 7](#_Toc181966325)

[2.5. List of registers 8](#_Toc181966326)

[2.6. Registers Description 8](#_Toc181966327)

[3. Implementation of module IP\_SMBus 10](#_Toc181966328)

[3.1. Submodule WB\_ITF 10](#_Toc181966329)

[Description of submodule WB\_ITF 10](#_Toc181966330)

[Functional description of submodule WB\_ITF 10](#_Toc181966331)

[Submodule WB\_ITF interface description: 10](#_Toc181966332)

[3.2. Submodule SMBus\_Controller 12](#_Toc181966333)

[Description of submodule SMBus\_Controller 12](#_Toc181966334)

[Functional description of submodule SMBus\_Controller 12](#_Toc181966335)

[Submodule SMBus\_Controller interface description: 14](#_Toc181966336)

[4. Simulations: 15](#_Toc181966337)

List of figures

[Figure 1 : Architecture of IP\_SMBus 6](#_Toc181966338)

[Figure 2 : Functional Diagram of IP\_SMBus 6](#_Toc181966339)

[Figure 3 : Description of WB\_ITF 10](#_Toc181966340)

[Figure 4 : Description of SMBus\_Controller 12](#_Toc181966341)

[Figure 5 : Finite State Machine of SMBus\_Controller 13](#_Toc181966342)

[Figure 6 : Write Command timing 15](#_Toc181966343)

[Figure 7 : Read Command timing 15](#_Toc181966344)

List of tables

[Table 1 : Module IP\_SMBus I/O Interface 7](#_Toc181966345)

[Table 2 : Memory mapping of IP\_SMBus 8](#_Toc181966346)

[Table 3 : CONTROL Register details 8](#_Toc181966347)

[Table 4 : STATUS Register details 8](#_Toc181966348)

[Table 5 : ADDRESS Register details 9](#_Toc181966349)

[Table 6 : TRANSMIT Register details 9](#_Toc181966350)

[Table 7 : RECEIVE Register details 9](#_Toc181966351)

[Table 8 : Submodule WB\_ITF I/O Interface 10](#_Toc181966352)

[Table 9 : Submodule SMBus\_Controller I/O Interface 14](#_Toc181966353)

# Introduction

The System Management Bus (SMBus) is a two-wire interface through which simple system and power management related chips can communicate with the rest of a system. SMBus provides a control bus for system and power management related tasks. The SMBus is a multi-master bus, meaning that more than one device capable of controlling the bus can be connected to it. This core is based on the SMBus 2.0 specification.

## Presentation of the document

The System Management Bus Core is SMBus 2.0 compliant, ensuring compatibility with the latest System Management Bus standards. The device supports all SMBus Address Resolution Protocol (ARP) commands and allows for an assignable SMBus slave address, providing flexibility in addressing. Additionally, it handles both SMBus reset commands and supports SMBus clock synchronization and clock stretching for efficient communication. Furthermore, it is compatible with the WISHBONE Rev B.3 System-on-Chip (SOC) bus standard, making it a versatile and robust solution for various applications.

## Principle and application domain

* Principle

SMBus operates on a master-slave architecture, where the master device initiates communication and the slave devices respond. It supports multiple devices on the same bus, with each device having a unique address. Key features include clock synchronization, clock stretching, and error checking to ensure reliable data transfer.

* Application Domain

SMBus is widely used in computer systems and embedded applications for tasks such as:

* Power Management: Managing power supplies, battery charging, and monitoring battery status.
* System Monitoring: Reporting system health, temperature, voltage, and fan speed.
* Device Configuration: Setting parameters for various components like memory modules and sensors.
* Error Reporting: Communicating error conditions and system status to the host controller.

Overall, SMBus is essential for efficient system management and communication between various hardware components.

## Targeted hardware

# Global description of module IP\_SMBus

## Architecture of module IP\_SMBus

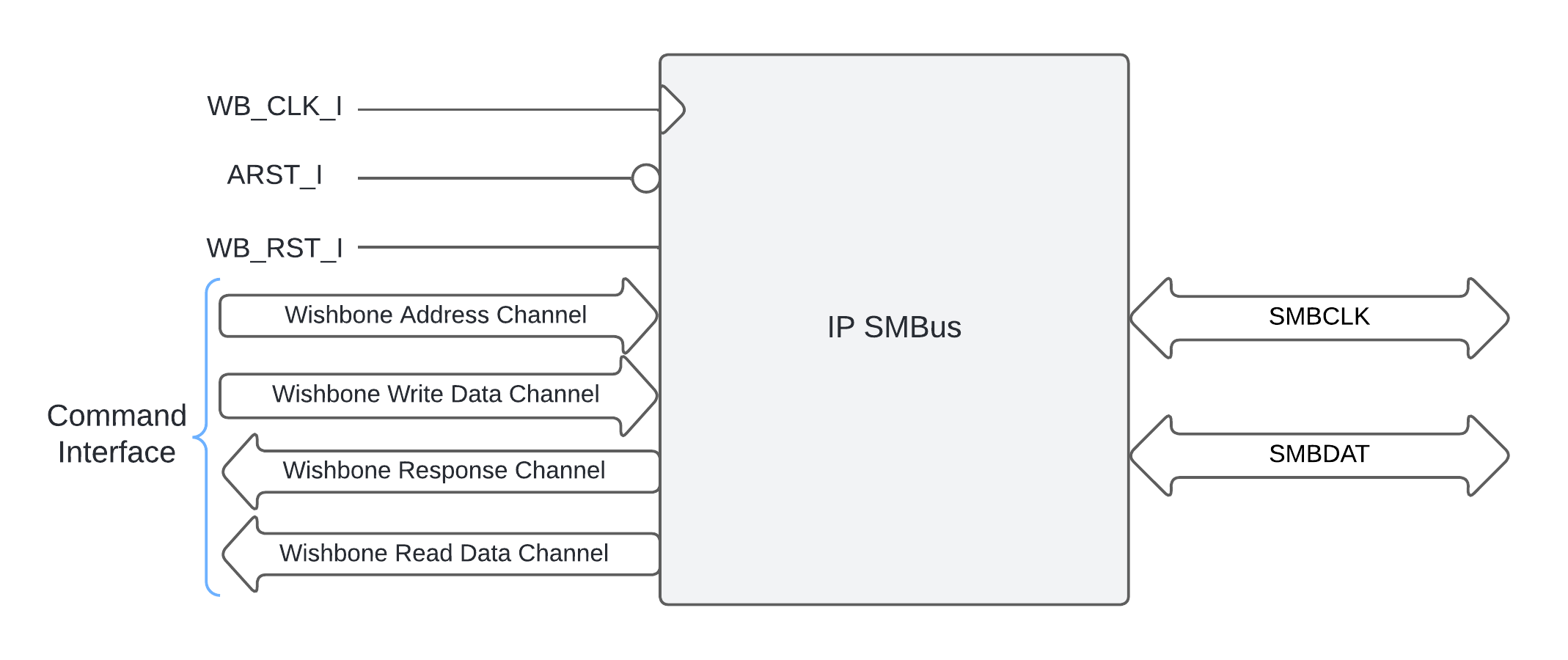


Figure 1 : Architecture of IP\_SMBus

* **SMBus Controller**:
  + Supports SMBus 2.0 protocol
  + Master and Slave modes
  + Clock stretching and arbitration
  + Configurable clock frequency
* **Wishbone Interface**:
  + Compliant with Wishbone B3 specification
  + Supports 8 data transfers
  + Interrupt support

## Functional diagram of module IP\_SMBus

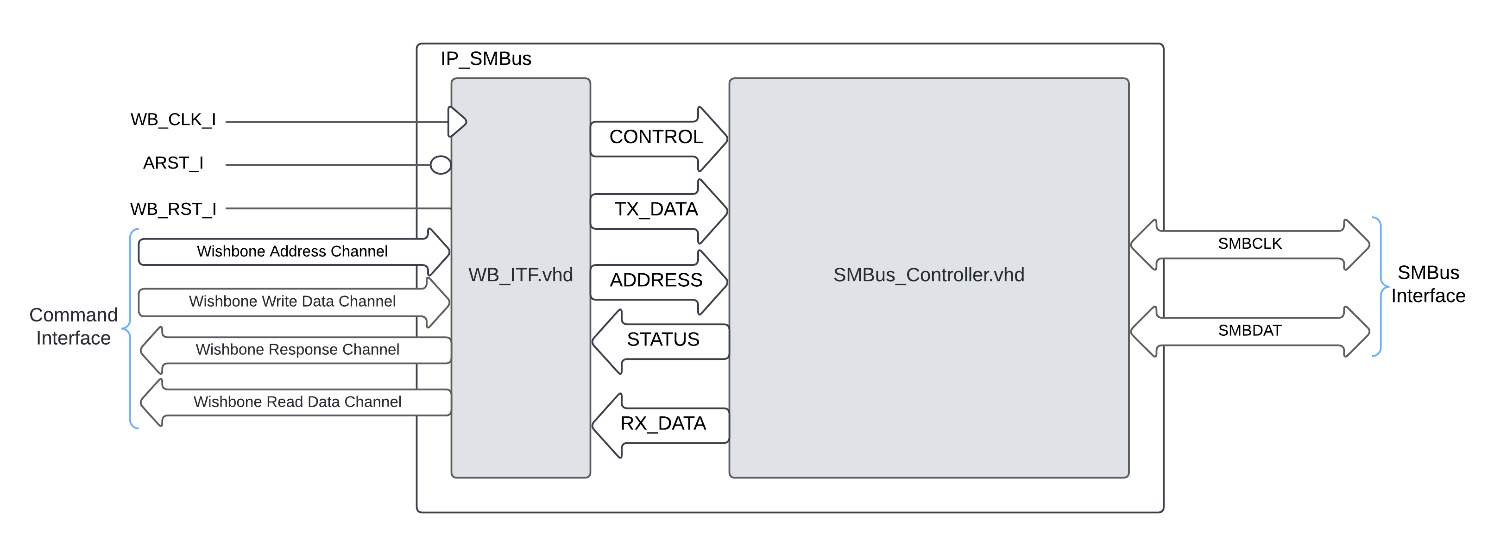


Figure 2 : Functional Diagram of IP\_SMBus

## Specifications of module IP\_SMBus

This IP\_SMBUS have 2 parameters :

* INPUT\_CLK : Which is the FPGA clock
* BUS\_CLK : Which is SMBCLK frequence target (between 100kHz and 10kHz)

## Module IP\_SMBus interface description (IO Ports)

Table 1 : Module IP\_SMBus I/O Interface

|  |  |  |
| --- | --- | --- |
| **Generic Parameters** | INPUT\_CLK | FPGA clock input : Default value : 100 MHz |
| BUS\_CLK | SMBus clock target : Default value : 100 kHz |
| **Inputs** | WB\_CLK\_I | Wishbone clock input |
| WB\_RST\_I | Wishbone synchronous reset input |
| ARST\_I | Wishbone asynchronous reset input |
| WB\_ADR\_I | 3bits Wishbone address input |
| WB\_DAT\_I | 8Bits Wishbone data input |
| WB\_WE\_I | Wishbone write enable input |
| WB\_STB\_I | Wishbone strobe input |
| WB\_CYC\_I | Wishbone valid bus cycle input |
| SMBCLK | SMBus bi-directional clock |
| SMBDAT | SMBus bi-directional data |
| **Outputs** | WB\_DAT\_O | 8bits Wishbone data output |
| WB\_ACK\_O | Wishbone acknowledge output |
| WB\_INTA\_O | Wishbone interrupt output |
| SMBCLK | SMBus bi-directional clock |
| SMBDAT | SMBus bi-directional data |

## List of registers

Table 2 : Memory mapping of IP\_SMBus

|  |  |  |  |
| --- | --- | --- | --- |
| Address Offset \* | Name | Access | Description |
| 00h | CONTROL | R/W | Control Register |
| 01h | STATUS | RO | Status Register |
| 02h | ADDRESS | WO | Device Address Register |
| 03h | TRANSMIT | WO | Transmit data Register |
| 04h | RECEIVE | RO | Receive data Register |

\*Notes : Address offset is relative to IP SMBus base address

## Registers Description

* CONTROL (Control register Address – Offset 00h)

Table 3 : CONTROL Register details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bits | Fields Name | Default Value | Access Type | Description |
| 0 | EN | 0 | R/W | Enable signal to start a transaction |
| 1 | RWB | 0 | R/W | Read/Write bit :   * 0 : Write command * 1 : Read command |
| 2 | TR\_s | 0 | R/W | Enable successive transaction |
| 7 to 3 | Reserved | 0 | N/A | N/A |

* STATUS (Status register Address – Offset 01h)

Table 4 : STATUS Register details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bits | Fields Name | Default Value | Access Type | Description |
| 0 | BUSY | 0 | RO | SMBus Controller Busy signals :   * 0 : Controller in IDLE State * 1 : Controller transmit or receive data |
| 1 | RXACK | 0 | RO | Acknowledge signal from Slave Device from SMBDAT :   * 0 : Slave Acknowledge * 1 : Slave NOT Acknowledge |
| 7 to 2 | Reserved | 0 | N/A | N/A |

* ADDRESS (Address register Address – Offset 02h)

Table 5 : ADDRESS Register details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bits | Fields Name | Default Value | Access Type | Description |
| 7 to 0 | ADDR | 0 | WO | Device Address to transmit on SMBDAT |

* TRANSMIT (Transmit register Address – Offset 03h)

Table 6 : TRANSMIT Register details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bits | Fields Name | Default Value | Access Type | Description |
| 7 to 0 | TX\_DATA | 0 | WO | Data to transmit on SMBDAT |

* RECEIVE (Receive register Address – Offset 04h)

Table 7 : RECEIVE Register details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bits | Fields Name | Default Value | Access Type | Description |
| 7 to 0 | RX\_DATA | 0 | RO | Data received from SMBDAT |

# Implementation of module IP\_SMBus



## Submodule WB\_ITF

### Description of submodule WB\_ITF

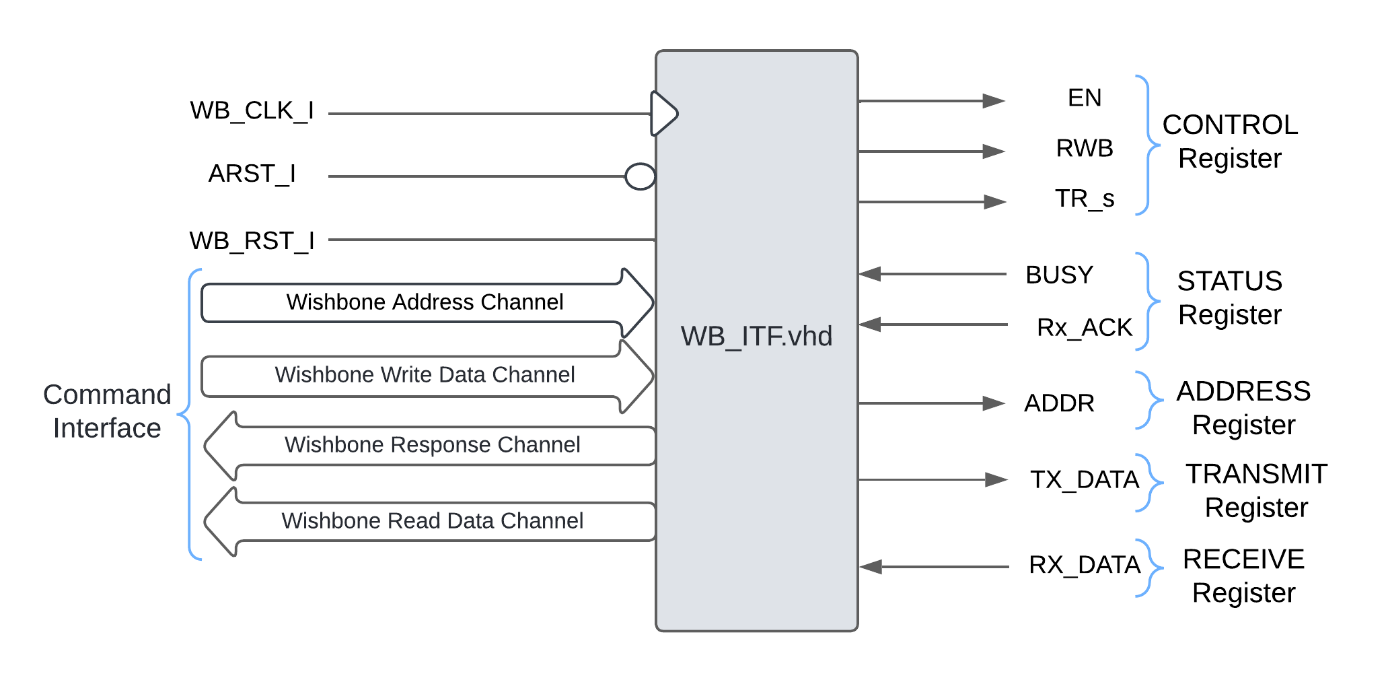


Figure 3 : Description of WB\_ITF

### Functional description of submodule WB\_ITF

This sub-modules integrate the internal registers. These are available with read and write access through WB\_WE\_I signal. The features of WB\_ITF are listed below :

* Reset Handling: Manages asynchronous and synchronous resets.
* Address Decoding: Uses address inputs to determine which internal register to read from or write to.
* Data Transfer: Facilitates data transfer between the Wishbone bus and internal registers.
* Control and Status Registers: Manages control signals and status reporting for the interface.

This module is designed to interface with other components in a system-on-chip (SoC) environment, providing a standardized communication protocol for data exchange.

### Submodule WB\_ITF interface description:

Table 8 : Submodule WB\_ITF I/O Interface

|  |  |  |
| --- | --- | --- |
| **Inputs** | WB\_CLK\_I | Wishbone clock input |
| WB\_RST\_I | Wishbone synchronous reset input |
| ARST\_I | Wishbone asynchronous reset input |
| WB\_ADR\_I | 3bits Wishbone address input |
| WB\_DAT\_I | 8Bits Wishbone data input |
| WB\_WE\_I | Wishbone write enable input |
| WB\_STB\_I | Wishbone strobe input |
| WB\_CYC\_I | Wishbone valid bus cycle input |
| Busy | Busy signal from SMBus Controller for STATUS Register |
| RxACK | Slave Device Acknowlegde from SMBus Controller for STATUS Register |
| RX\_DATA | 8bits Received data from SMBus Controller for RECEIVE Register |
| **Outputs** | WB\_DAT\_O | 8bits Wishbone data output |
| WB\_ACK\_O | Wishbone acknowledge output |
| WB\_INTA\_O | Wishbone interrupt output |
| ADDR | 7bits Address Slave Device from ADDRESS Register for SMBus Controller |
| EN | Enable transaction from CONTROL Register for SMBus Controller |
| RWB | Read/Write bit command from CONTROL Register for SMBus Controller |
| TR\_S | Enable successive transaction from CONTROL Register for SMBus Controller |
| TX\_DATA | 8bits Data to transmit on SMBDAT from TRANSMIT Register for SMBus Controller |

## Submodule SMBus\_Controller

### Description of submodule SMBus\_Controller

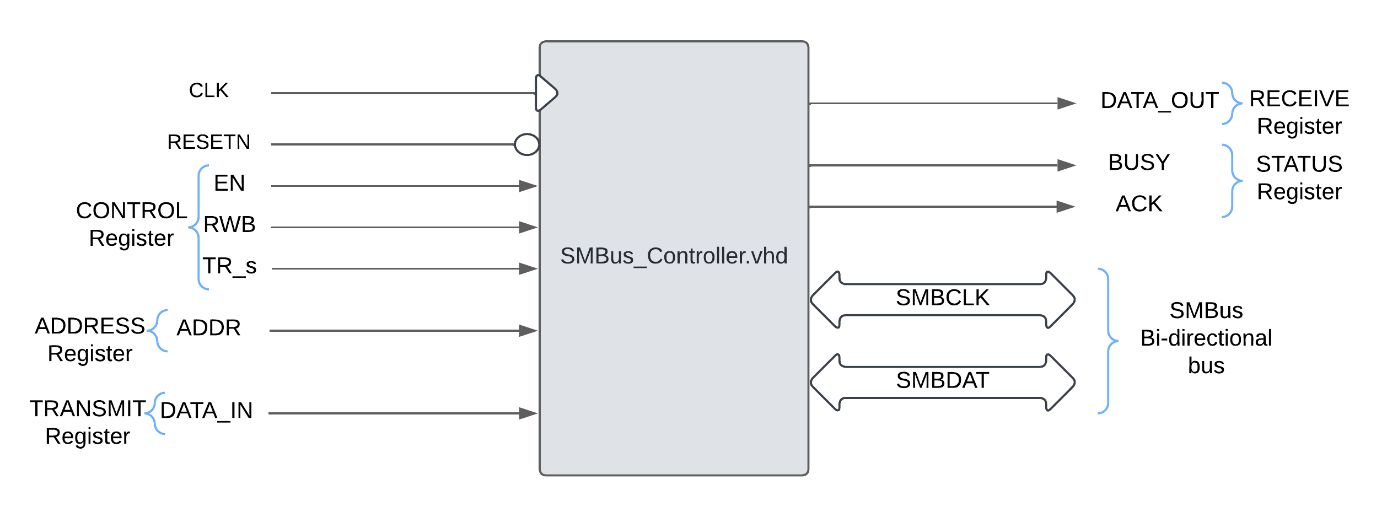


Figure 4 : Description of SMBus\_Controller

### Functional description of submodule SMBus\_Controller

In this module, the SMBus clock is generated with a process. The process is designed to generate specific timing signals based on an input clock and a reset signal. It uses a counter to keep track of clock cycles, counting up to a value determined by a multiplier. When the reset signal is active, the process initializes certain control signals and resets the counter.

On each rising edge of the clock, the process updates a previous clock state and increments the counter if a specific condition is met. The process then uses a series of conditions to set the values of the timing signals based on the counter’s value.

During the first quarter of the cycle, both timing signals are low. In the second quarter, one of the timing signals goes high. In the third quarter, the other timing signal goes high, and a control signal is set based on another condition.

In the final quarter, the first timing signal goes low again while the second remains high. This ensures the generation of synchronized timing signals with specific characteristics.

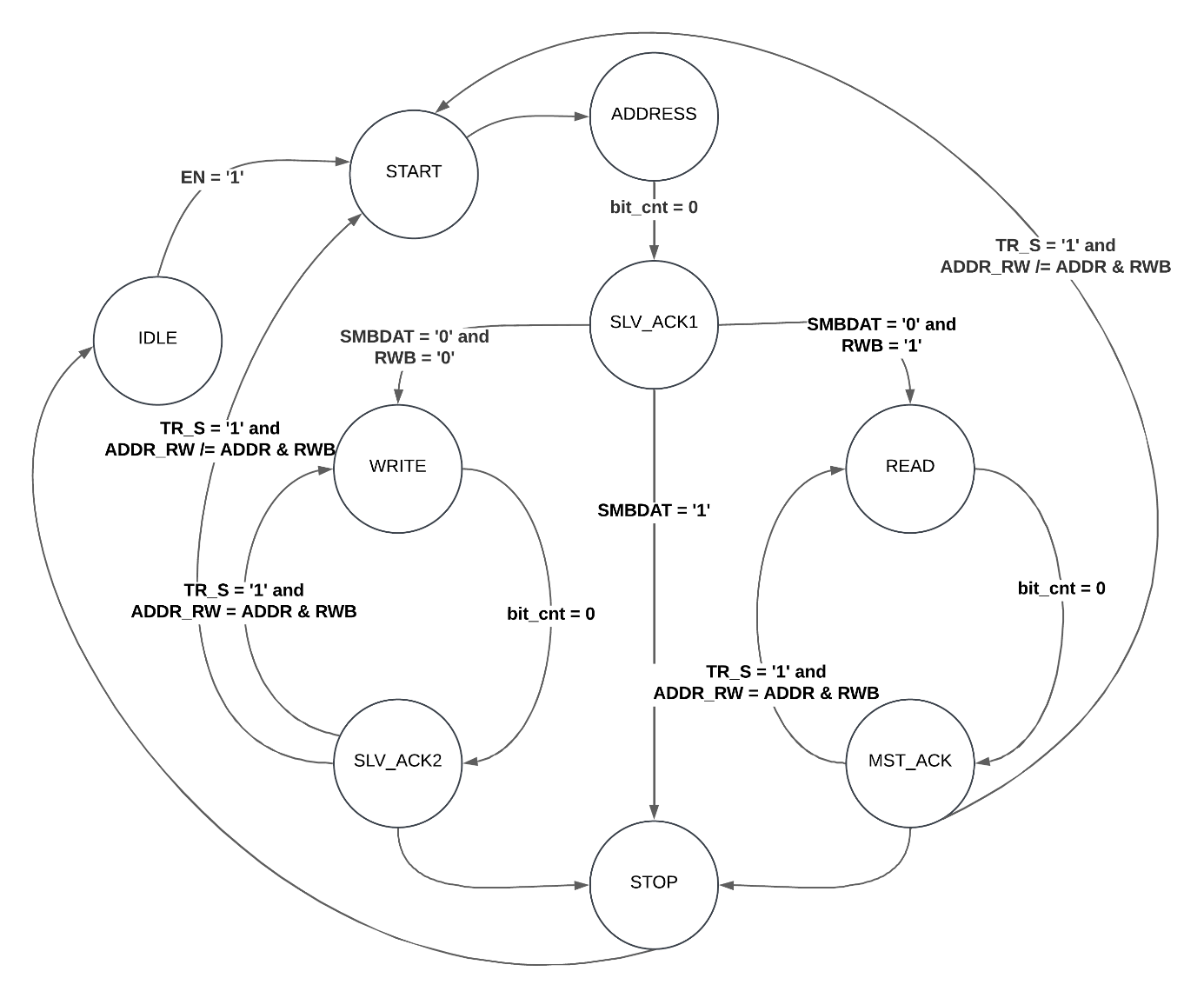
After generating the clock for the SMBus transaction, the next step of the module uses a Finite State machine (FSM) that generates SMBDAT from the signals available in the internal registers.

Figure 5 : Finite State Machine of SMBus\_Controller

The transition between states are written in the figure 6 and are made on a rising edge of the FPGA clock:

* State IDLE : Initialize all signals to be on a stable state
* State START : Write ‘Z’ on SMBDAT to start a transaction
* State ADDRESS : Write ‘Z’ or ‘0’ on SMBDAT with the help of a shift register operation
* State SLV\_ACK1 : Wait acknowledge of Slave device
* State WRITE : if RWB = ‘0’ we make a write command with following data from TRANSMIT register
* State READ : if RWB = ’1’, receive the data from SMBDAT and push the data in the RECEIVE register
* State SLV\_ACK2 : After a write transaction wait for Slave device acknowledge
* State MST\_ACK : After a read transaction the SMBus Controller make acknowledge on SMBDAT
* State STOP : Write ‘Z’ on SMBDAT to describe the end of a SMBus transaction

### Submodule SMBus\_Controller interface description:

Table 9 : Submodule SMBus\_Controller I/O Interface

|  |  |  |
| --- | --- | --- |
| **Generic Parameters** | INPUT\_CLK | FPGA clock input : Default value : 100 MHz |
| BUS\_CLK | SMBus clock target : Default value : 100 kHz |
| **Inputs** | CLK | Wishbone clock input |
| RESETN | Wishbone asynchronous reset input |
| EN | Enable transaction from WB\_ITF |
| RWB | Read/Write bit command from WB\_ITF |
| TR\_S | Enable successive transaction from WB\_ITF |
| ADDR | 7bits Address Slave Device from WB\_ITF |
| DATA\_IN | 8bits Data to transmit on SMBDAT from WB\_ITF |
| SMBCLK | SMBus bi-directional clock |
| SMBDAT | SMBus bi-directional data |
| **Outputs** | BUSY | Busy signal for WB\_ITF |
| ACK | Slave Device Acknowlegde for WB\_ITF |
| DATA\_OUT | 8bits Received data for WB\_ITF |
| SMBCLK | SMBus bi-directional clock |
| SMBDAT | SMBus bi-directional data |

# Simulations:

* SMBus Write Command timing

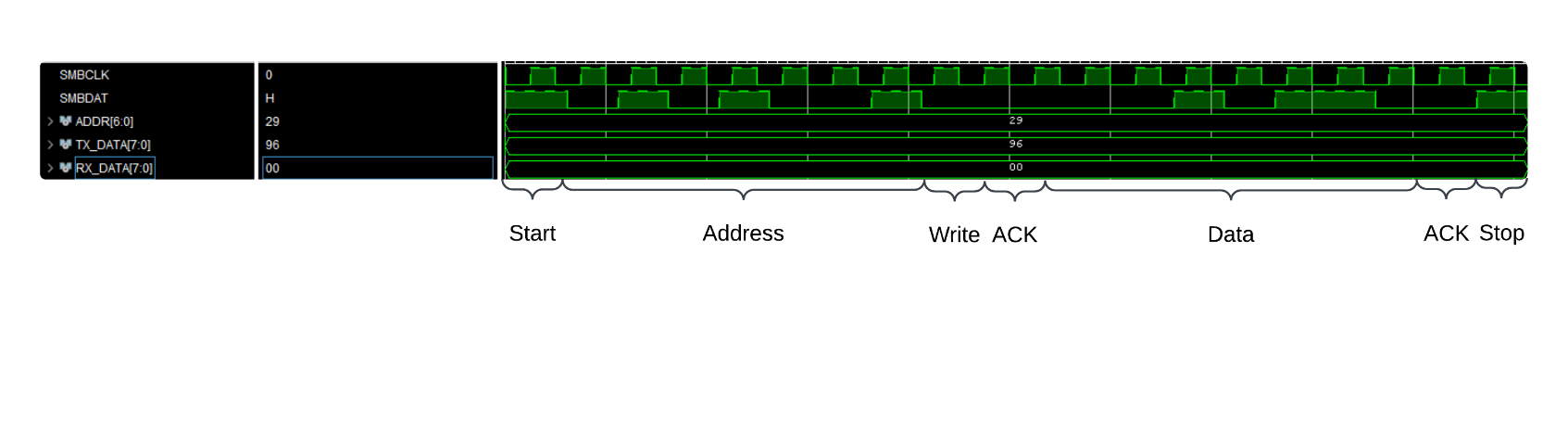


Figure 6 : Write Command timing

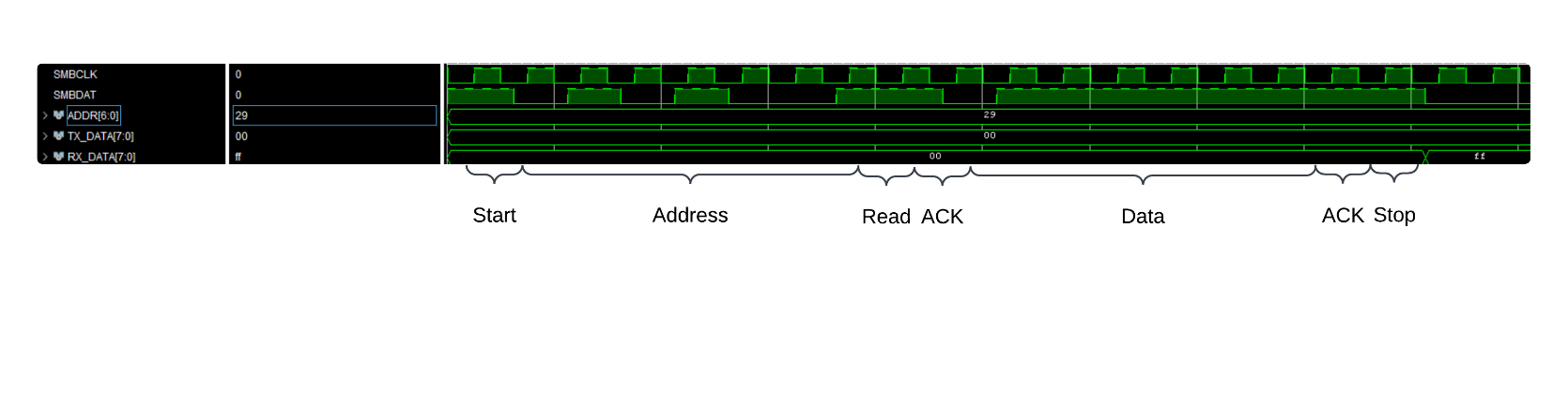
* SMBus Read Command timing

Figure 7 : Read Command timing