



Fraunhofer Institut
Integrierte Schaltungen

***CorePool* FHG_I2C**

Databook

- subject to change without notice -

Version: v1.4

Date: 06.08.02

Document History

Table 1: Document History

Version	Date	Responsible	Description
v1.0	16.01.96	Kam	Generic datasheet
v1.1	12.08.97	Shu	
v1.2	26.08.97	Shu	
v1.3	28.04.98	Shu	Update document structure, use asynchronous instead of synchronous reset
v1.4	09.07.98	Shu	Add Fast Mode Clock Ratio (6:10)

Contact

CorePool
Fraunhofer Institute Integrated Circuits

Am Wolfsmantel 33
91058 Erlangen
Germany

Phone: +49 (0) 9131 776 777
Fax: +49 (0) 9131 776 499
Email: info@corepool.com
Internet: <http://www.corepool.com>

Table of Contents

Document History	2
Contact	2
Purpose	7
Features	7
Design Kit	7
Requirements	8
Block Diagram	9
Signal Description	10
Parameter Description	13
VHDL Usage trough Component Instantiation	14
Verilog Usage trough Component Instantiation	17
Functional Description	19
Application Interface	21
Timing Diagrams	28
Application Notes	37

List of Tables and Figures

Document History	2
Block Diagram	9
I2C-Bus Interface Signals to the System	10
I2C-Bus Interface Signals to the I2C-Bus	10
I2C-Bus Interface Signals to the Master-Application	10
I2C-Bus Interface Signals to the Slave-Application	11
Parameter Description	13
Functional Parameters	24
Addressing Parameter	24
Bus-Clock Frequency Parameter	25
Dependence of Bus-Clock Frequency on System-Clock Frequency	26
Standard-Mode	26
Fast_Modes	27
Legend to the Timing-Diagrams	28
Master-Transmitter - Slave-Receiver: Start of a Frame (7Bit Address)	28
Master-Transmitter - Slave-Receiver: Start of a Frame (10 Bit Address)	29
Master-Receiver - Slave-Transmitter: Start of a Frame (7Bit Address)	29
Master-Receiver - Slave-Transmitter: Start of a Frame (10 Bit Address)	30
Master-Transmitter - Slave-Receiver: Master Terminates Frame	30
Master-Transmitter - Slave-Receiver: Slave Terminates Frame	31
Master-Transmitter - Slave-Receiver: Slave Terminates Frame, Master-Applikation Reacts with Delay	32
Master-Receiver - Slave-Transmitter: Master Terminates Frame	33
Master-Transmitter - Slave-Receiver: Master Terminates Job	33
Addressing of a Disabled Slave and End of Job	34
Delay Due to Late MASTER_DATAIN_VAL	34
Delay Due to Late SLAVE_DATAOUT_ACC	35
Eventual Handshaking Speed-Up Due to Premature DATAIN_VAL and/or DATAOUT_ACC	35
Start Byte	36
Master Application	38
Slave Application	39

Purpose

The I²C-bus interface serves as a bidirectional on-chip communication interface between parallel-bus applications and the I²C-bus. The interface implements the I²C-bus protocol for master and slave applications.

The concept of the I²C-bus is to provide the whole functionality of the I²C-bus protocol to the user keeping the interface communication with the application as simple as possible. The user interface enables the application to communicate with the I²C-bus on a parallel address and data port controlled by handshaking.

The I²C-bus interface is parameterizable in its functionality, address mode and transfer rate. Thus, the interface can link a master and slave application to the I²C-bus both as receiver and transmitter, supports 7 bit or 7/10 bit addressing and the transfer mode *standard* or *fast* with transfer rates of 100 kbit/s up to 400 kbit/s respectively.

These functional parameters are constants which instantiate the required components and functions as hardware when the I²C-bus is synthesized. Thereby the functional parameterizability allows an application specific implementation without hardware overlap.

The I²C-bus interface disposes of discrete data ports for each functional block. The synchronization of the data flow with the application is done bitwise by handshaking. The storage of transfer data and addresses must be done by the application in order to insure that no data will be lost in the I²C-interface due to transfer errors.

Features

- parallel interface to master and slave applications with handshaking
- parameterizable functionality (master/slave receiver/transmitter),
- parameterizable address mode (7/10 bit)
- parameterizable transfer rate (100-400 kbit/s)
- separate ports for each functional block
- no data storage, which insures uncorrupted data in case of transfer errors
- cell area of approximately 600 to 1800 gate equivalents depending on the interface functionality

Design Kit

- Technology Independent Implementation as Synopsys Design Ware Components
- VHDL/Verilog Simulation Models
- VHDL/Verilog Compliance Test Suite
- Auxiliary Simulation Models for User Testbenches
- Synthesis and Testsynthesis Scripts

- Example Design and Testchip available
- Design Support, Netlist Synthesis Service and Consulting available

Requirements

Simulation

- VHDL IEEE-1076 Simulator
- Verilog IEEE-1364 Simulator

Synthesis

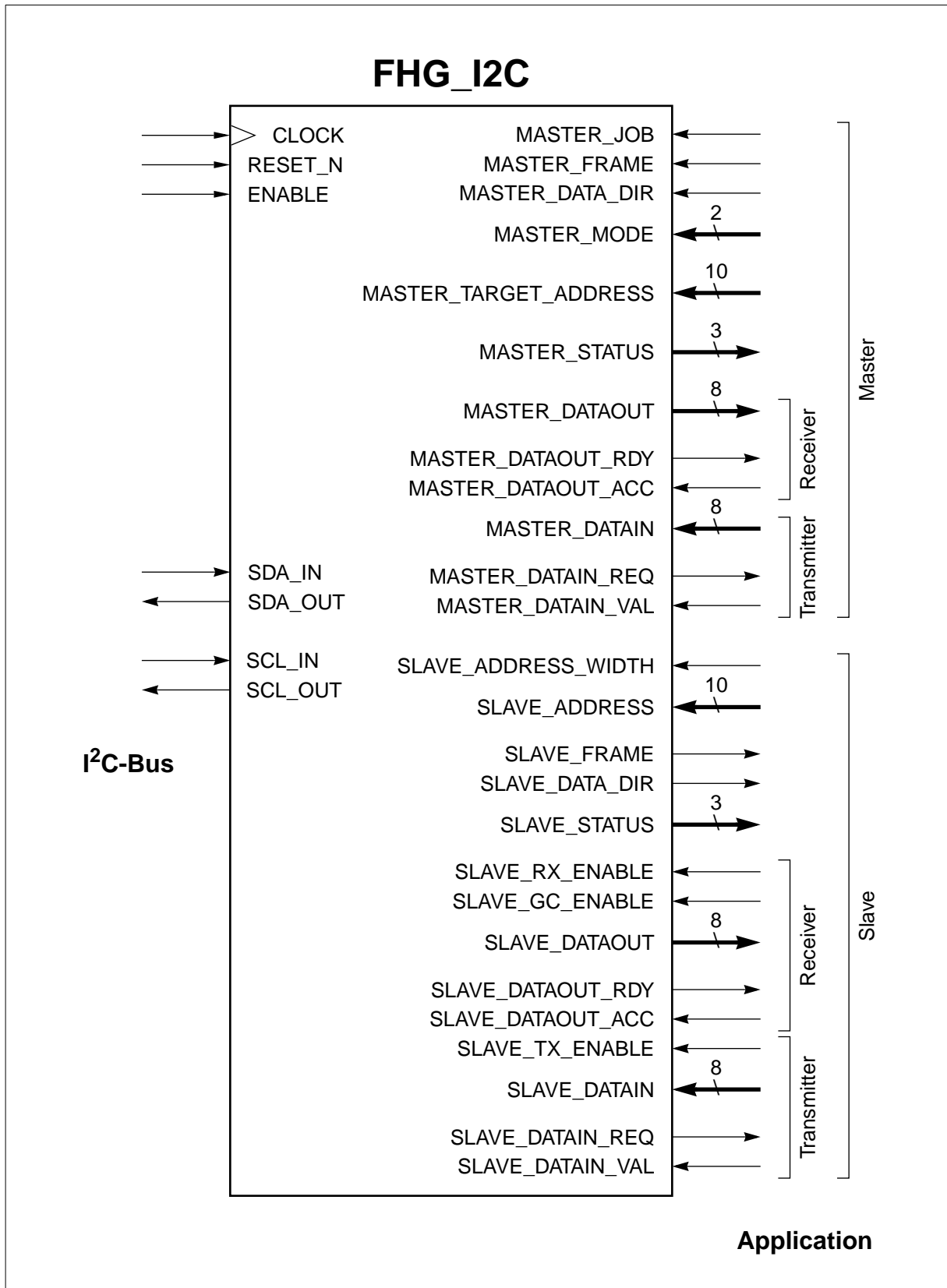
- Synopsys Design Compiler

References

- Philips Semiconductors, *The I²C-bus and how to use it (including specifications)*, April 1995

Block Diagram

Figure 1: Block Diagram



Signal Description

Table 2: I²C-Bus Interface Signals to the System

Signal	I / O	Bits	Purpose
CLOCK	in	1	system clock
RESET_N	in	1	asynchronous reset (active low)
ENABLE	in	1	chip select

Table 3: I²C-Bus Interface Signals to the I²C-Bus

Signal	I / O	Bits	Purpose
SCL_IN	in	1	bus clock input
SCL_OUT	out	1	bus clock output
SDA_IN	in	1	bus data input
SDA_OUT	out	1	bus data output

Table 4: I²C-Bus Interface Signals to the Master-Application

Signal	I / O	Bits	Purpose
MASTER_JOB	in	1	determines at the end of a frame whether the bus will be released (= 0) or the next frame will be issued (= 1)
MASTER_FRAME	in	1	determines the beginning and the end of a frame
MASTER_DATA_DIR	in	1	data flow direction: 0: data transmission 1: data request
MASTER_MODE	in	2	transmit: 00: to 7 bit address 01: to 10 bit address 10: as General Call 11: START Byte request: x0: from 7 bit address x1: from 10 bit address

Table 4: I²C-Bus Interface Signals to the Master-Application

Signal	I / O	Bits	Purpose
MASTER_STATUS	out	3	000: BUS_FREE 001: ADDRESSING1 (1st byte) 010: ADDRESSING2 (2nd byte) 011: SENDING 100: RECEIVING 101: EOF (end of frame) 110: ARB_LOST (arbitration lost) 111: BUSY (bus not free)
MASTER_TARGET_ADDRESS	in	10	slave address to be responded
MASTER_DATAIN	in	8	data from the application to the master
MASTER_DATAIN_REQ	out	1	request to the application to supply a new data byte (request)
MASTER_DATAIN_VAL	in	1	acknowledge of the application that a new data byte has been provided (valid)
MASTER_DATAOUT	out	8	data from the master to the application
MASTER_DATAOUT_RDY	out	1	request to the application to accept a new data byte (ready)
MASTER_DATAOUT_ACC	in	1	acknowledge of the application that the new data byte has been accepted (accept)

Table 5: I²C-Bus Interface Signals to the Slave-Application

Signal	I / O	Bits	Purpose
SLAVE_TX_ENABLE	in	1	slave transmit enable
SLAVE_RX_ENABLE	in	1	slave receive enable
SLAVE_GC_ENABLE	in	1	slave general call receive enable
SLAVE_FRAME	out	1	slave receives or transmits data
SLAVE_DATA_DIR	out	1	data flow direction: 0: transmit 1: receive
SLAVE_ADDRESS_WIDTH	in	1	slave address width: 0: 7 bit 1: 10 bit

Table 5: I²C-Bus Interface Signals to the Slave-Application

Signal	I / O	Bits	Purpose
SLAVE_ADDRESS	in	10	corresponding slave address
SLAVE_DATAIN	in	8	data from the application to the slave
SLAVE_DATAIN_REQ	out	1	request to the application to supply a new data byte (request)
SLAVE_DATAIN_VAL	in	1	acknowledge from the application that a new data byte has been provided (valid)
SLAVE_DATAOUT	out	8	data from the slave to the application
SLAVE_DATAOUT_RDY	out	1	request to the application to accept the new data byte (ready)
SLAVE_DATAOUT_ACC	in	1	acknowledge of the application that the new data byte has been accepted (accept)
SLAVE_STATUS	out	3	000: LISTEN (waiting for a start-condition) 001: ADDRESSING1 (1st byte) 010: ADDRESSING2 (2nd byte) 011: SENDING 100: RECEIVING 101: REC_GC (receiving data on general call)

Parameter Description

Table 6: Parameter Description

Name	Type	Default Value	Value Range	Description
FUNCTION_MASTER_RECEIVER	integer	0	{0,1}	implementation of the corresponding functional block
FUNCTION_MASTER_TRANSMITTER	integer	0	{0,1}	
FUNCTION_SLAVE_RECEIVER	integer	0	{0,1}	
FUNCTION_SLAVE_TRANSMITTER	integer	0	{0,1}	
ADDRESSING_MODE	integer	0	{0,1}	7bit or 7/10 bit
SPEED_MODE	integer	0	{0:3}	standard / fast mode

The chosen parameters are constants, which are hard implemented by the synthesis of the interface-module and are not changeable in operation. Due to these parameters, it is assured that only required functions are instantiated in hardware.

VHDL Usage through Component Instantiation

The following VHDL-code depicts a simple example for the instantiation of the FHG_I2C module as a master-receiver/transmitter being 7 and 7/10 bit addressable with a transfer rate of up to 100 kbit/s.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

library DW_FHG_I2C;
use DW_FHG_I2C.I2C_IF_COMPONENTS.all;

entity MY_DESIGN is
  port (
    -- globals
    CLOCK: in          std_logic;
    RESET_N          : in  std_logic;
    ENABLE           : in  std_logic;
    -- i2c-bus port
    SDA_IN           : in  std_logic;
    SDA_OUT          : out std_logic;
    SCL_IN           : in  std_logic;
    SCL_OUT          : out std_logic;
    -- master-application port
    MASTER_JOB       : in  std_logic;
    MASTER_FRAME     : in  std_logic;
    MASTER_DATA_DIR  : in  std_logic;
    MASTER_MODE      : in  std_logic_vector(1 downto 0);
    MASTER_TARGET_ADDRESS : in  std_logic_vector(9 downto 0);
    MASTER_STATUS    : out std_logic_vector(2 downto 0);
    MASTER_DATAOUT   : out std_logic_vector(7 downto 0);
    MASTER_DATAOUT_RDY : out std_logic;
    MASTER_DATAOUT_ACC : in  std_logic;
    MASTER_DATAIN    : in  std_logic_vector(7 downto 0);
    MASTER_DATAIN_REQ : out std_logic;
    MASTER_DATAIN_VAL : in  std_logic);
end MY_DESIGN;

architecture STRUCTURE of MY_DESIGN is

  signal ZERO          : STD_LOGIC;
  signal ZERO_BYTE    : STD_LOGIC_VECTOR (7 downto 0);
  signal ZERO_ADDRESS : STD_LOGIC_VECTOR (9 downto 0);

begin
```

```
ZERO          <= `0`;
ZERO_BYTE     <= (others => `0`);
ZERO_ADDRESS  <= (others => `0`);

I2C_INTERFACE : FHG_I2C
generic map (
    FUNCTION_MASTER_RECEIVER    => 1,    -- implemented
    FUNCTION_MASTER_TRANSMITTER => 1,    -- implemented
    FUNCTION_SLAVE_RECEIVER     => 0,    -- not implemented
    FUNCTION_SLAVE_TRANSMITTER  => 0,    -- not implemented
    ADDRESSING_MODE              => 1,    -- 7 and 7/10 bit
    SPEED_MODE                   => 0)    -- standard
port map (
    -- globals
    CLOCK          => CLOCK,
    RESET_N        => RESET_N,
    ENABLE          => ENABLE,
    -- i2c-bus port
    SDA_IN         => SDA_IN,
    SDA_OUT        => SDA_OUT,
    SCL_IN         => SCL_IN,
    SCL_OUT        => SCL_OUT,
    -- master-application port
    MASTER_JOB     => MASTER_JOB,
    MASTER_FRAME   => MASTER_FRAME,
    MASTER_DATA_DIR => MASTER_DATA_DIR,
    MASTER_MODE    => MASTER_MODE,
    MASTER_TARGET_ADDRESS => MASTER_TARGET_ADDRESS,
    MASTER_STATUS  => MASTER_STATUS,
    MASTER_DATAOUT => MASTER_DATAOUT,
    MASTER_DATAOUT_RDY => MASTER_DATAOUT_RDY,
    MASTER_DATAOUT_ACC => MASTER_DATAOUT_ACC,
    MASTER_DATAIN  => MASTER_DATAIN,
    MASTER_DATAIN_REQ => MASTER_DATAIN_REQ,
    MASTER_DATAIN_VAL => MASTER_DATAIN_VAL,
    -- slave application port (not implemented)
    SLAVE_ADDRESS_WIDTH => ZERO,
    SLAVE_ADDRESS       => ZERO_ADDRESS,
    SLAVE_FRAME         => open,
    SLAVE_DATA_DIR     => open,
    SLAVE_STATUS       => open,
    SLAVE_RX_ENABLE    => ZERO,
    SLAVE_GC_ENABLE    => ZERO,
    SLAVE_TX_ENABLE    => ZERO,
    SLAVE_DATAOUT      => open,
    SLAVE_DATAOUT_RDY  => open,
    SLAVE_DATAOUT_ACC  => ZERO,
    SLAVE_DATAIN       => ZERO_BYTE,
```

```
SLAVE_DATAIN_REQ    => open ,  
SLAVE_DATAIN_VAL    => ZERO) ;
```

```
end STRUCTURE ;
```

Verilog Usage through Component Instantiation

The following VHDL-code depicts a simple example for the instantiation of the FHG_I2C module as a master-receiver/transmitter being 7 and 7/10 bit addressable with a transfer rate of up to 100 kbit/s.

```
module MY_DESIGN (  
    // globals  
    CLOCK,  
    RESET_N,  
    ENABLE,  
    // i2c-bus port  
    SDA_IN,  
    SDA_OUT,  
    SCL_IN,  
    SCL_OUT,  
    // master-application port  
    MASTER_JOB,  
    MASTER_FRAME,  
    MASTER_DATA_DIR,  
    MASTER_MODE,  
    MASTER_TARGET_ADDRESS,  
    MASTER_STATUS,  
    MASTER_DATAOUT,  
    MASTER_DATAOUT_RDY,  
    MASTER_DATAOUT_ACC,  
    MASTER_DATAIN,  
    MASTER_DATAIN_REQ,  
    MASTER_DATAIN_VAL);  
  
    parameter FUNCTION_MASTER_RECEIVER    = 1; //implemented  
    parameter FUNCTION_MASTER_TRANSMITTER = 1; //implemented  
    parameter FUNCTION_SLAVE_RECEIVER     = 0; //not implemented  
    parameter FUNCTION_SLAVE_TRANSMITTER  = 0; //not implemented  
    parameter ADDRESSING_MODE             = 0; //7 and 7/10 bit  
    parameter SPEED_MODE                   = 0; //standard  
  
    // globals  
    input          CLOCK;  
    input          RESET_N;  
    input          ENABLE;  
    // i2c-bus port  
    input          SDA_IN;  
    output         SDA_OUT;  
    input          SCL_IN;
```

```
output          SCL_OUT;
// master-application port
input           MASTER_JOB;
input           MASTER_FRAME;
input           MASTER_DATA_DIR;
input[1:0]      MASTER_MODE;
input[9:0]      MASTER_TARGET_ADDRESS;
output[2:0]     MASTER_STATUS;
output[7:0]     MASTER_DATAOUT;
output          MASTER_DATAOUT_RDY;
input           MASTER_DATAOUT_ACC;
input[7:0]      MASTER_DATAIN;
output          MASTER_DATAIN_REQ;
input           MASTER_DATAIN_VAL;

FHG_I2C #(
    FUNCTION_MASTER_RECEIVER,
    FUNCTION_MASTER_TRANSMITTER,
    FUNCTION_SLAVE_RECEIVER,
    FUNCTION_SLAVE_TRANSMITTER,
    ADDRESSING_MODE,
    SPEED_MODE)
I2C_INTERFACE (
    // globals
    .CLOCK(CLOCK),
    .RESET_N(RESET_N),
    .ENABLE(ENABLE),
    // i2c-bus port
    .SDA_IN(SDA_IN),
    .SDA_OUT(SDA_OUT),
    .SCL_IN(SCL_IN),
    .SCL_OUT(SCL_OUT),
    // master-application port
    .MASTER_JOB(MASTER_JOB),
    .MASTER_FRAME(MASTER_FRAME),
    .MASTER_DATA_DIR(MASTER_DATA_DIR),
    .MASTER_MODE(MASTER_MODE),
    .MASTER_TARGET_ADD(MASTER_TARGET_ADDRESS),
    .MASTER_STATUS(MASTER_STATUS),
    .MASTER_DATAOUT(MASTER_DATAOUT),
    .MASTER_DATAOUT_RD(MASTER_DATAOUT_RDY),
    .MASTER_DATAOUT_AC(MASTER_DATAOUT_ACC),
    .MASTER_DATAIN(MASTER_DATAIN),
    .MASTER_DATAIN_REQ(MASTER_DATAIN_REQ),
    .MASTER_DATAIN_VAL(MASTER_DATAIN_VAL));
```

```
endmodule
```

Functional Description

Logic

- high = 1 = supply voltage
- low = 0 = ground

Naming Conventions

The I²C-bus is primarily conceived for the transfer of control information between solitary Integrated Circuits in a system. It consists of the two bidirectional lines *SDA* (serial data) and *SCL* (serial clock). These are connected to the supply voltage by pull-up resistors. The output stages of the devices connected to the I²C-bus have to be open-collector or open-drain. Thus, the bus lines perform the wired-AND function. The number of devices connected to the same I²C-bus is limited only by a maximum bus capacitance of 400 pF.

Mode

There are two ranges specified for the bus-clock frequency:

- *standard-mode* up to 100 kHz,
- *fast-mode* up to 400 kHz.

Function

- The *master* starts and terminates a data transfer and generates the bus-clock.
- The *slave* is addressed by the master and synchronizes itself with the master by the bus-clock.
- The *transmitter* sends data to the receiver via the bus
- The *receiver* receives data via the bus from the transmitter.

Both master and slave can act as transmitter and/or receiver.

Arbitration

If there is more than one master connected to the bus, it might be that two or more masters start a transfer simultaneously. In order not to disturb each other, an *arbitration* procedure is implemented to control the masters priorities.

The master always has to listen to his own transfer. Once the received character differs from the one transmitted, the master which wanted to transmit a $SDA = 1$ has to abort its transfer. Due to the wired-AND functionality of the bus it is ensured that the winning master can continue its transfer undisturbed.

Bus-Free, START- and STOP-Condition, Data, Acknowledge

In “stand-by“ mode, the two bus lines are kept at high-level by pull-up resistors (*bus-free*).

If the condition *bus-free* is valid, the master starts its transfer with an event which can be distinguished from normal data, the *START-condition*. The transfer is terminated by the definite *STOP-condition*. If the master wants to terminate a transfer and start another following without releasing the bus, he might issue another *START-condition* instead of a *STOP-condition* (*re-*

peated START-condition).

Between START- and STOP-condition, data (addresses, real data and acknowledge) are sent on the bus. The data on the SDA line must be stable (valid) during the high period of the bus-clock. The high or low state of the data line can only change when the bus-clock SCL is low.

The continuation of the transfer after each data word (8 bit) is signalled by the receiver with an *acknowledge* (SDA low).

SCL Low

Each slave has the possibility to interrupt the transfer on the bus. Thereto, the bus-clock SCL has to be pulled on low-level. If the master detects a $SCL = 0$ even though he sent a $SCL = 1$, he has to insert wait cycles until the slave releases the clock line again ($SCL = 1$) and thus, the sent and detected level on the clock-line are equal. This mechanism is based on the wired-AND functionality of the I²C-bus as well.

Job and Frame

Job signifies an order of the application to the master to communicate with one or more slaves. The master allocates the I²C-bus during the entire process.

Frame signifies the addressing phase and the following data transfer phase (only in one direction). A frame covers everything on the I²C-bus between a START-condition and the following START-condition (or STOP-condition if the end of job has been reached).

If the master has to change its data flow direction during communication with a certain slave, he will have to use two separate frames, one after the other.

A job might consist of one or more frames.

Application Interface

General Aspects

The concept of the I²C-bus is to provide the whole functionality of the I²C-bus protocol to the user, keeping the interface communication with the application as simple as possible. The interface enables the application to communicate with the I²C-bus on a parallel address and data port controlled by handshaking.

The storage of data and addresses has to be done by the application. The I²C-bus interface contains no latches except the shift-register for the parallel/serial conversion of transfer data. Therefore no data can be corrupted due to transfer errors.

Requirements

System-Clock

The system clocks of the application and the interface have to be synchronous to each other, which means that they derive from a common clock. Different clocks may be used, since all data is transferred by a handshake mechanism. In general this premise would be fulfilled if the application and the I²C-bus interface are integrated together on one chip. If not, all interface input signals must be synchronized by the application of at least one flip-flop clocked with the interface's system-clock. Thereby the interface's reaction to signals of the application is delayed.

Pads

At the bus line inputs SCL and SDA schmitt-triggers have to occur. The output-stages have to be built as open-collector or open-drain stages and be capable of ensuring 1000 ns (standard-mode) / 300 ns (fast mode) rise-time and 300 ns fall-time with a maximum bus line capacitance of 400 pF. The output-stages have to show a certain slew-rate and be *floating* when the supply voltage is turned off. The input-value of the pad has to be mapped concurrently to its output (*monitoring*).

Protocol

Transfer Initialization

A transfer is initiated by the master-application setting MASTER_FRAME = 1 and MASTER_JOB = 1, thus asking the master to transmit or receive data. If the bus is already occupied by another master, its request will be idle until the bus is free again. When the bus is freed, the request is served at once. With every acknowledge, the signals MASTER_FRAME and MASTER_JOB are sampled. The transfer goes on as long as MASTER_FRAME = 1 is true. In this case, the value of MASTER_JOB is not relevant.

Transfer Termination

Only the master-application is able to terminate a frame directly, by setting MASTER_FRAME = 0. Then, the frame will be terminated after the current byte.

The slave-application is only able to terminate a frame indirectly as a receiver, by sending a *not-acknowledge*. This triggers the end of transfer in the master-interface. This is indicated to the master-application by MASTER_STATUS = EOF (*End Of Frame*). Then the application

has to terminate the frame directly by sending `MASTER_FRAME = 0`.

A slave-application as transmitter can not terminate a frame. This has to be considered in the overall protocol.

When the master terminates a frame he has the possibility to start a new frame at once without releasing the bus. The signal `MASTER_JOB` at the end of a frame is responsible for the deallocation of the bus. If `MASTER_JOB = 1` is true, the frame will be terminated at once and the next one will be started (*repeated START-condition*). Else if `MASTER_JOB = 0` is true, the frame and the job will be terminated (*STOP-condition*), thus releasing the bus.

Addresses

The I²C-bus protocol works with two different address-spaces. In case of a 7 bit address only one byte is used to build up the connection for a transfer, while in case of a 10 bit address two bytes are required. The address-space of a 7 bit address is restricted to addresses from (hex) 0x08 to 0x77 (112 different addresses). The remaining addresses are used for special purposes (e.g. to indicate a 10 bit address) or are reserved for future applications. The address-space of a 10 bit address is not restricted at all.

During the addressing-phase (first byte or the first two bytes after the *START-condition*) the address has to be constant. Like in the case of data, buffering has been omitted. Because the administration of the addresses has to be done by the application, it makes sense to employ storage there too.

Slave

The value at port `SLAVE_ADDRESS` depicts the address of the slave, by which he might be addressed directly.

The application-interface allows for separately switching on and off the communication for each data flow direction. For instance, a slave might be adjusted in a way, that he can be accessed as a receiver and not as a transmitter using the same target-address (perhaps because he doesn't have new data to send). For 7 bit addresses, this is applicable without restriction. If a slave-transmitter has to be accessed by a 10 bit address, the slave-receiver also has to be enabled (`SLAVE_TX_ENABLE = 1`, `SLAVE_RX_ENABLE = 1`). This behaviour has its explanation in the later expansion of the I²C-bus protocol to 10 bit addresses.

Master Target

The master hasn't got its own address, thus he isn't addressable. However he has to provide the address of the slave being addressed. This target-address must be stable at the port `MASTER_TARGET_ADDRESS` during the addressing-phase.

Data

Each of the four combinations of master/slave and receiver/transmitter possesses its own data-port, because data might come from different sources, or is determined for different sinks according to the functionality.

The buffering of data has been omitted knowingly, because the memory-architecture depends on the application very much and thus will be preferably implemented there. Also no data can be lost in the interface-module, because the next byte will only be requested by the transmitter-application (*request*), when the receiver-application has acknowledged the continuation of the transfer (*acknowledge*). Hence, it follows that the application must be able to react relatively quick, in order to not cause unnecessary delays.

Synchronization

The synchronization of the data-flow with the application occurs *bytewise* by two handshake-lines each. Each data-port has its own handshake-signals. Therefore, it is possible to connect, as an example, FIFOs as buffers in an easy way.

The handshake-signals haven't any influence on the beginning and ending of a transfer. They might cause transfer delays (by pulling SCL low), but can't stop the transfer.

In case of transfer blocked due to a missing masters $DATAIN_VAL = 1$ or $DATAOUT_ACC = 1$, the master-application must terminate the frame by sending $MASTER_FRAME = 0$. It is reasonable to provide a *timeout* in the master-application for this case.

If this occurs in a slave, the slave-receiver might send a *not-acknowledge* after the timeout to the master by assigning $SLAVE_RX_ENABLE = 0$, thus forcing him to terminate the frame.

The application signals $DATAIN_VAL$ and $DATAOUT_ACC$ might be assigned ($= 1$) prior to the request of the interface-module ($DATAIN_REQ$, $DATAOUT_RDY$), which might cause an acceleration of the handshaking.

General Call

A *general call* is a particular kind of a master-transmitter to slave-receiver transfer with a special 7 bit address (0x00). Therewith, it is possible, to access several slaves simultaneously. The slave has a special switch ($SLAVE_GC_ENABLE$), to control the acceptance of a general call. The to be sent data appears at the same output-port as the data being addressed directly. $SLAVE_STATUS$ indicates, that a general call is in progress.

Start Byte

The *start byte* has got the value (hex) 0x01 and is used to awake slow applications. It's intended for applications without a hardware-interface (e.g. directly connected microprocessors), which are sampling the SDA-line in relatively large time-steps. If a $SDA = 0$ is detected, the processor might switch to a higher sampling rate waiting for the start of the next frame.

The start-procedure (START-condition with following start-byte) represents a peculiar frame. This procedure will be repeated until the frame gets terminated by the master-application sending $MASTER_FRAME = 0$.

Parameters

Functional

Each combination of master/slave and receiver/transmitter might be chosen separately.

Table 7: Functional Parameters

Parameter	Value	Remark
FUNCTION_MASTER_RECEIVER	0 / 1	0: functionality won't be implemented
FUNCTION_MASTER_TRANSMITTER	0 / 1	
FUNCTION_SLAVE_RECEIVER	0 / 1	1: functionality will be implemented
FUNCTION_SLAVE_TRANSMITTER	0 / 1	

Addressing

The addressing-parameter is valid for the master (target address) and the slave (own address) respectively, in case of both being implemented due to the functional parameters.

Table 8: Addressing Parameter

Addressing Mode	Value	Remark
7 bit only	0	a 7 bit address is only valid in the interval from 0x08 to 0x77 (decimal: 8 to 119)
7 and 10 bit	1	

For the master a parameter value of 0 means, that only the lower 7 bits of the address-port MASTER_TARGET_ADDRESS are used as target-address. The upper 3 bits have to be set to constants. In this case, the choice of a 10 bit address by MASTER_MODE is not applicable and 7 bit addresses are used instead.

For the slave, a parameter value of 0 means that only the lower 7 bits of the address-port, SLAVE_ADDRESS, are used. The upper 3 bits have to be set to constants.

Transfer Rate

The I²C-bus specification defines two different transfer rate ranges:

- *Standard-Mode* at up to 100 kbit/s,
- *Fast-Mode* at up to 400 kbit/s.

For the *fast-mode*, downwards-compatibility is required, which means that fast-mode interfaces have to be able to communicate with standard-mode interfaces. The standard-mode interfaces slow down the fast-mode interfaces to 100 kHz bus-clock frequency by assigning SCL low in each bus cycle. This is only possible when the standard-mode slaves are able to detect a START-condition sent in fast-mode. Thus the standard-mode interfaces have to dispose of the corresponding sampling-rate as well.

Due to the ability to stop the transfer by assigning SCL low, the I²C-bus specification does not permit the external generation of the bus-clock SCL. The transfer-rate and thus, the bus-clock frequency is directly coupled to the system-clock frequency of the interface-module. The sampling-frequency is identical to the system-clock frequency. The valid range of the system-clock frequency is restricted by the following two conditions:

- *Sample Rate:* The lower boundary of the system-clock frequency is determined by the sampling and secure detection of the sent characters. This means, according to the presently employed digital input-filter, at least 3 samples have to appear within the minimal SCL-high time of the fast-mode (600 ns). From this follows a minimum sampling-time of 200 ns, according to a 5 MHz sampling-frequency.
- *Signal Generation:* Minimum SCL-high and -low times together with maximum rise- and fall-times naturally determine the upper boundary of the system-frequency.

In standard-mode, both requirements, together with the demand of as low as possible system-clock frequency, result in a definite system-clock frequency of 5 MHz.

In fast-mode, the system-clock frequency might be chosen between 5 MHz and 10 MHz. The maximum transfer-performance with a 400 kHz bus-clock can only be achieved at 10 MHz because here, the I²C-bus specifications claim a clock-ratio high:low of 9:16. In order to offer the user nearly the same performance even at low system-clock frequencies, two different modes might be chosen which differ from the duty cycle of the *generated* bus-clock. The sampling is done the same way in all modes (sample-clock = system-clock). Therefore, the speed-mode parameter is unnecessary for slave-interfaces.

Table 9: Bus-Clock Frequency Parameter

Speed Mode	Value	High:Low	Bus-Clock Frequency [kHz]	System-Clock Frequency [MHz]
standard	0	25:25	100	5
fast	1	6:10	312.5 - 390.6	5 - 6.25
	2	7:12	263.1 - 394.7	5 - 7.5
	3	9:16	200 - 400	5 - 10

The connected master-interfaces might be run in any mode. It is also unimportant, by which system-clock frequency the several interfaces are clocked. Attention must be paid that for each interface the valid range of the system-clock frequencies corresponding to the adjusted fast-mode are complied.

During the *arbitration* (two masters sending concurrently), the slower master determines the SCL-low time, whilst the faster one determines the SCL-high time.

Figure 2: Dependence of Bus-Clock Frequency on System-Clock Frequency

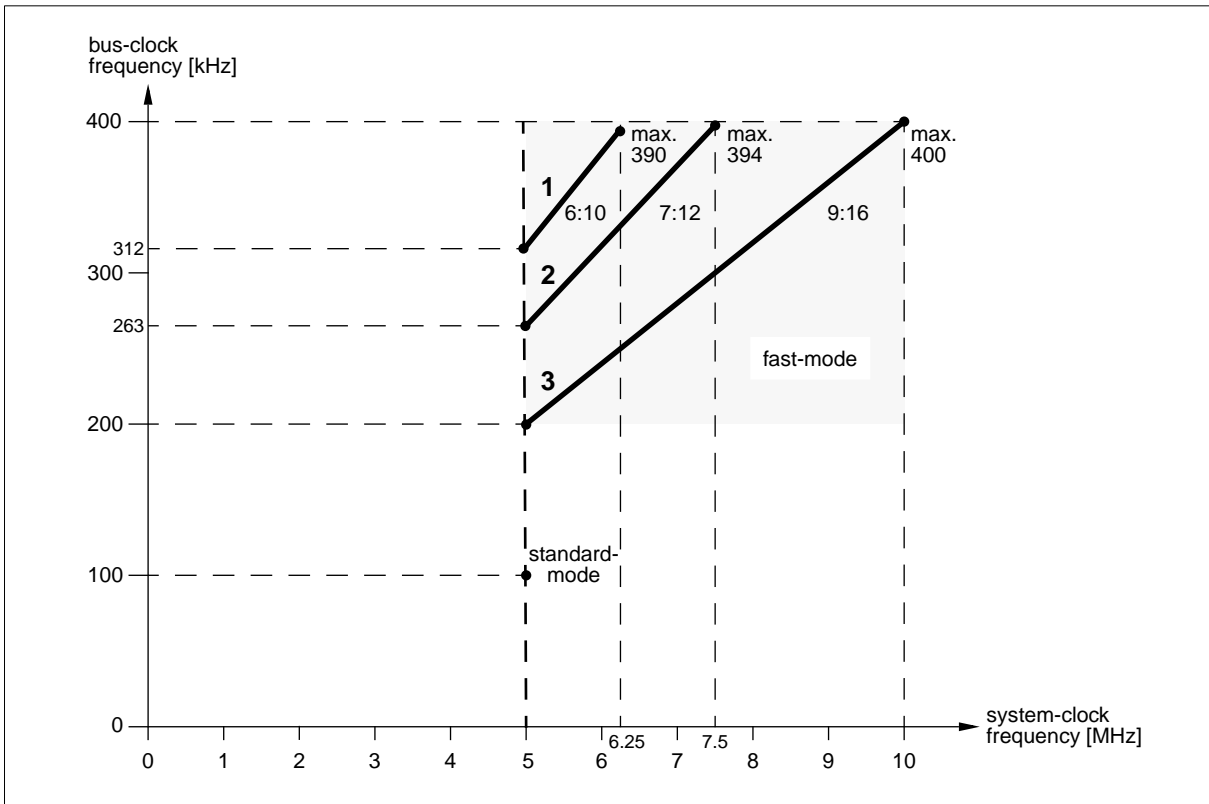


Figure 3: Standard-Mode

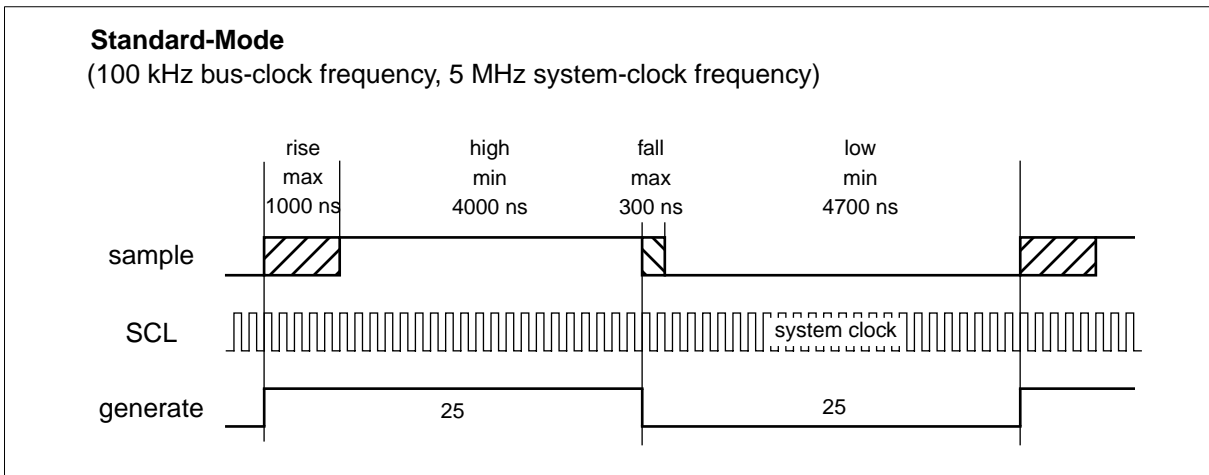
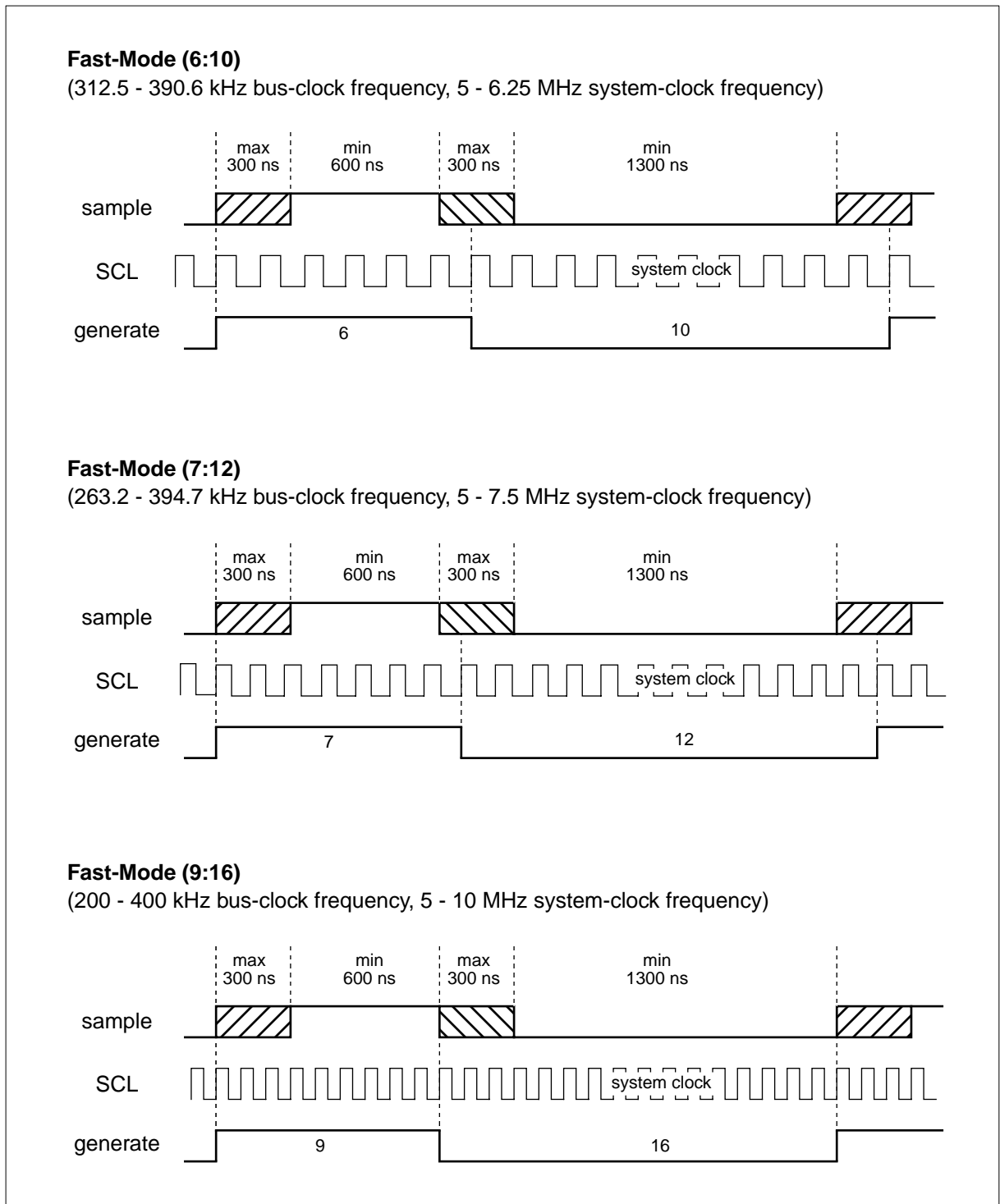


Figure 4: Fast_Modes



Timing Diagrams

Figure 5: Legend to the Timing-Diagrams

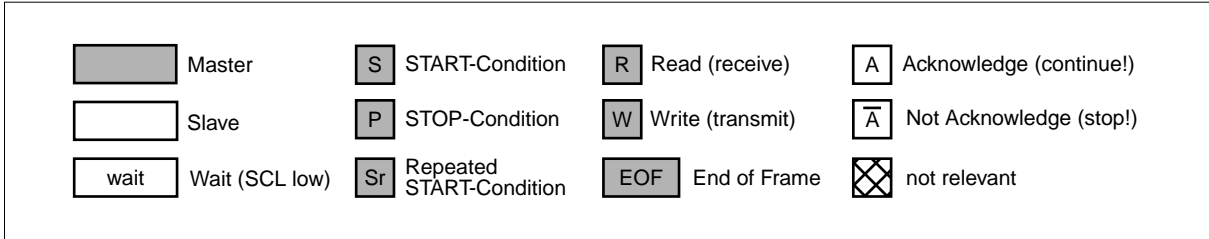


Figure 6: Master-Transmitter - Slave-Receiver: Start of a Frame (7Bit Address)

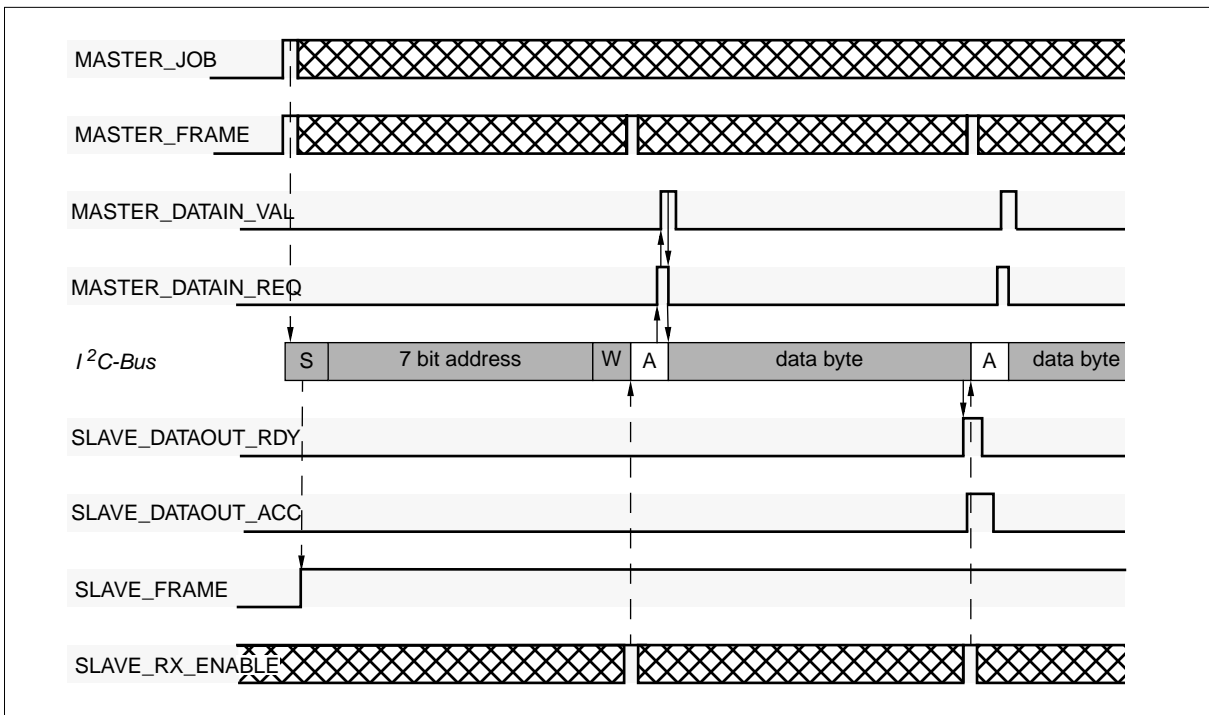


Figure 7: Master-Transmitter - Slave-Receiver: Start of a Frame (10 Bit Address)

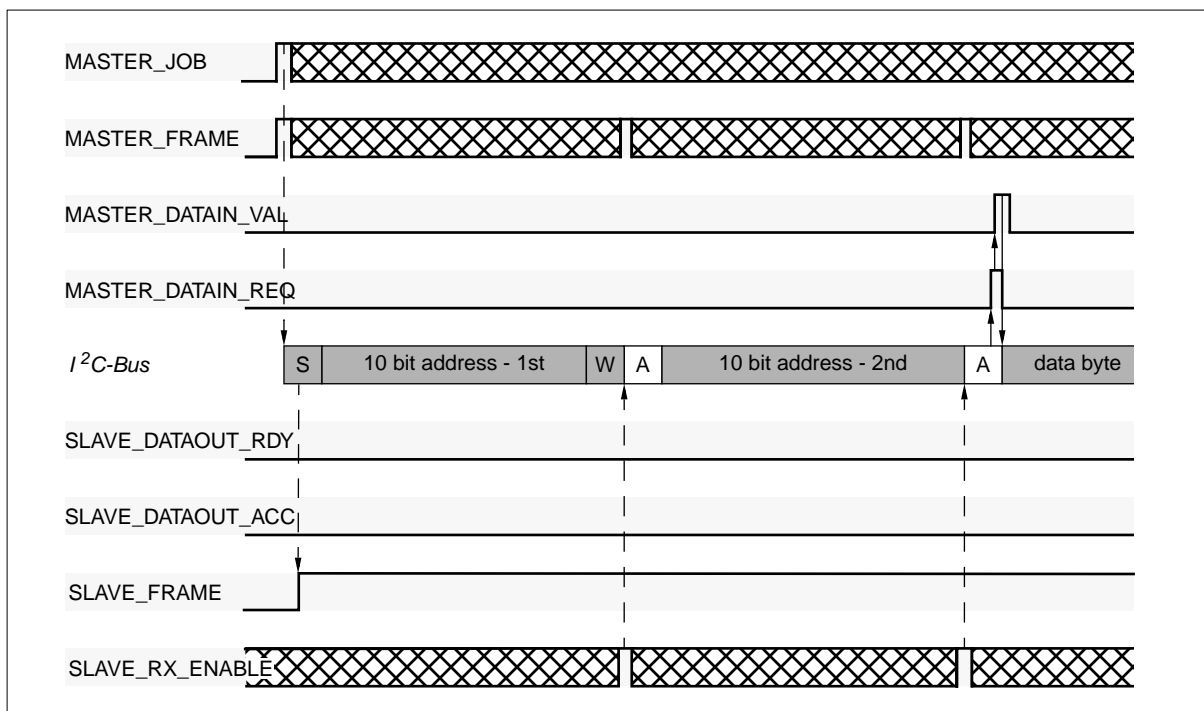


Figure 8: Master-Receiver - Slave-Transmitter: Start of a Frame (7Bit Address)

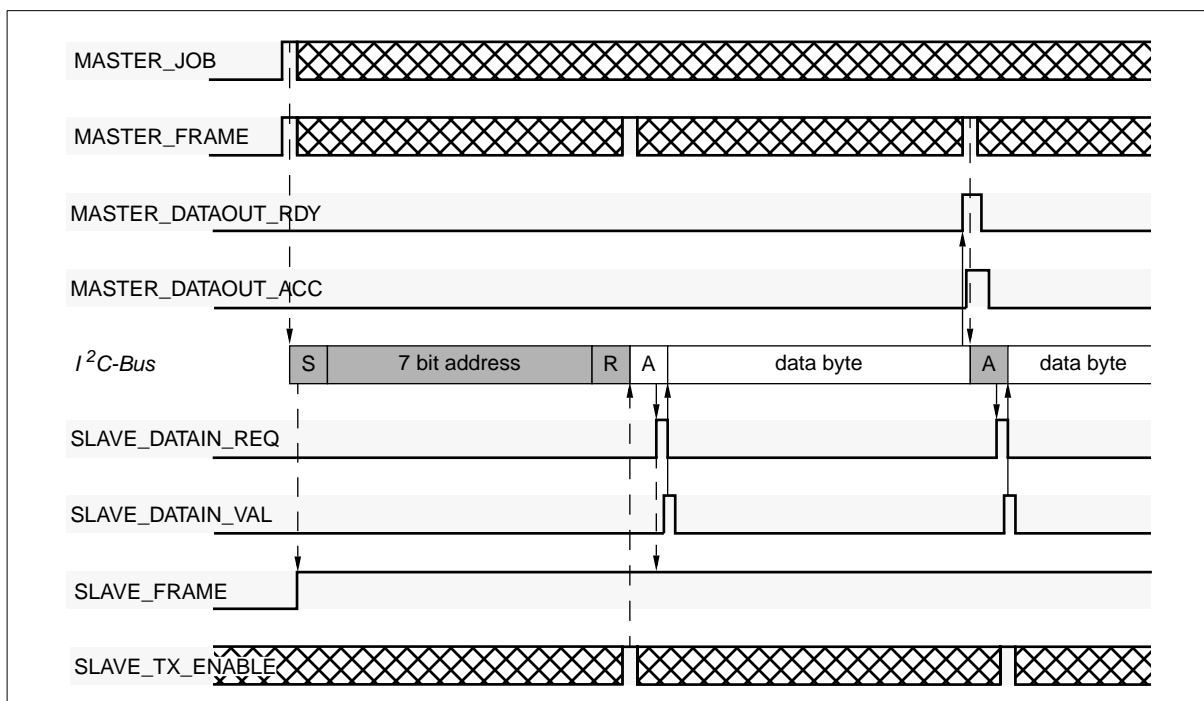


Figure 9: Master-Receiver - Slave-Transmitter: Start of a Frame (10 Bit Address)

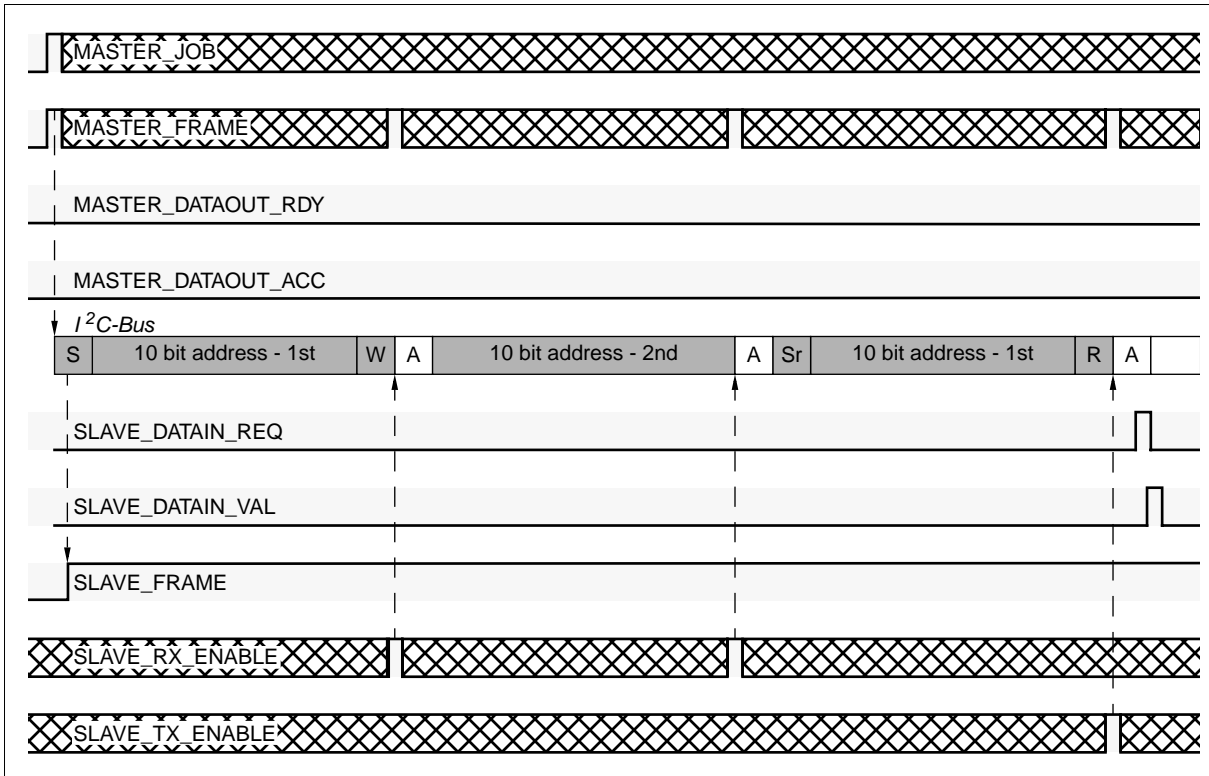


Figure 10: Master-Transmitter - Slave-Receiver: Master Terminates Frame

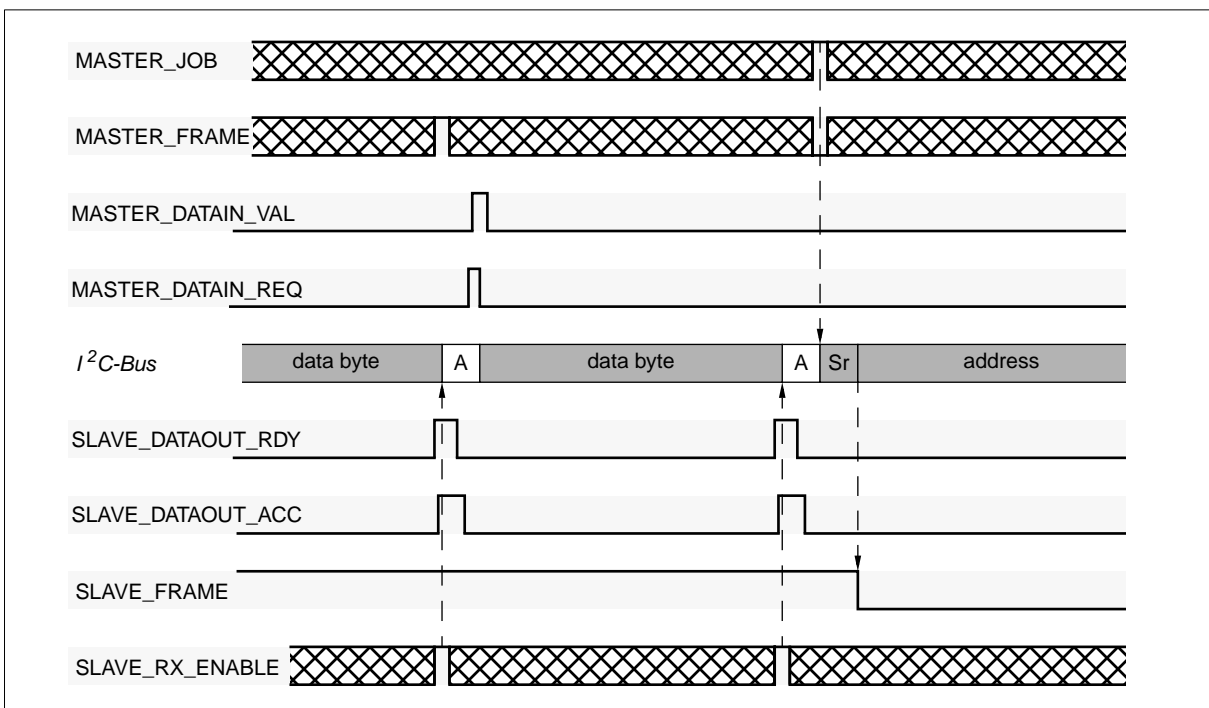


Figure 11: Master-Transmitter - Slave-Receiver: Slave Terminates Frame

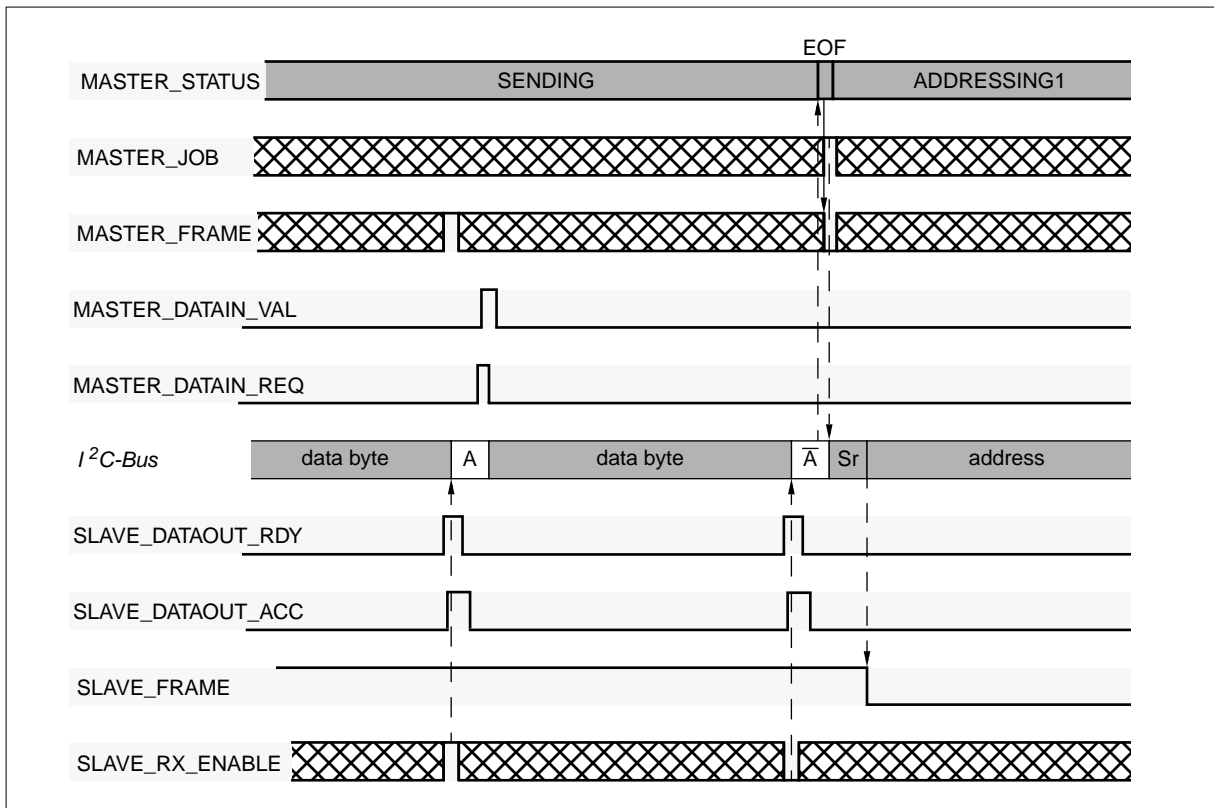


Figure 12: Master-Transmitter - Slave-Receiver: Slave Terminates Frame, Master-Applikation Reacts with Delay

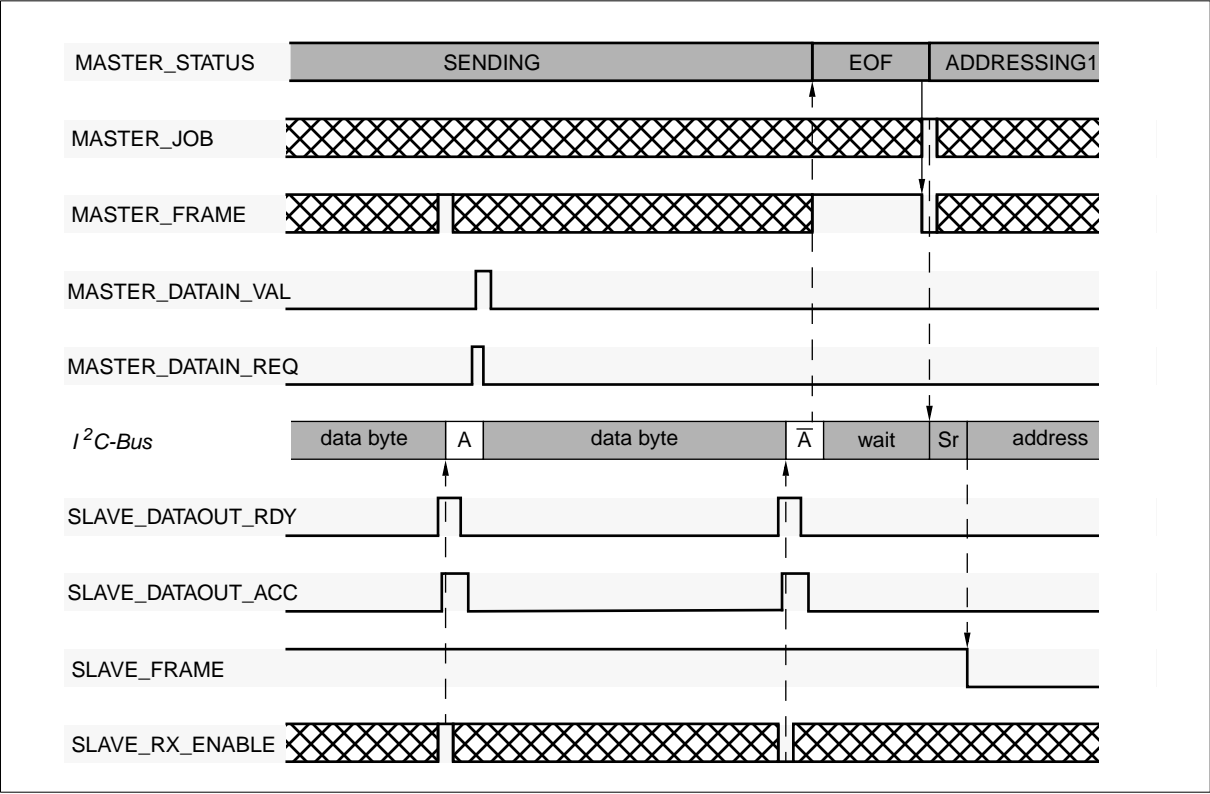


Figure 13: Master-Receiver - Slave-Transmitter: Master Terminates Frame

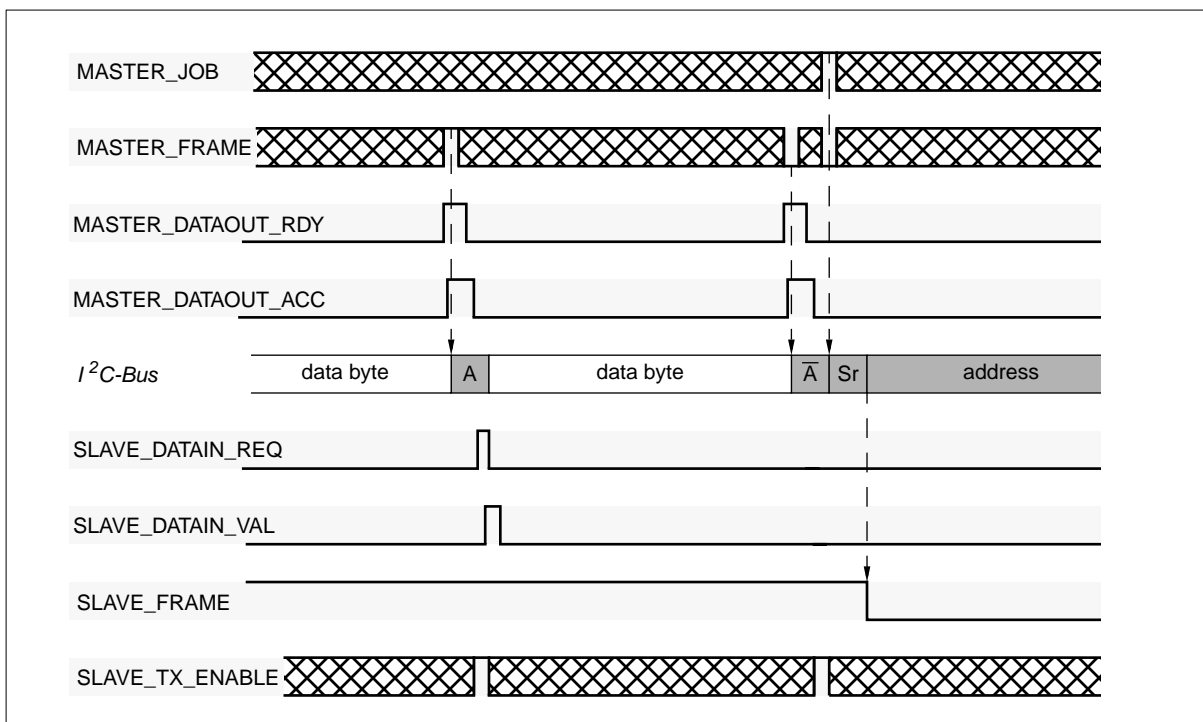


Figure 14: Master-Transmitter - Slave-Receiver: Master Terminates Job

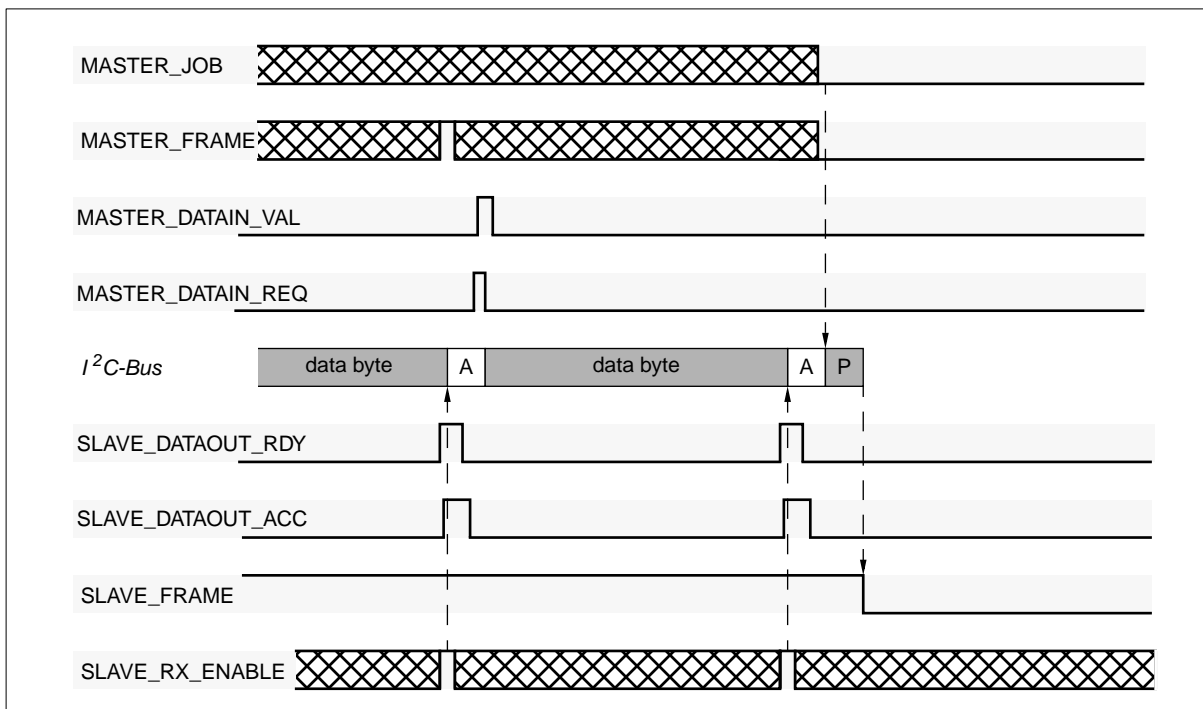


Figure 15: Addressing of a Disabled Slave and End of Job

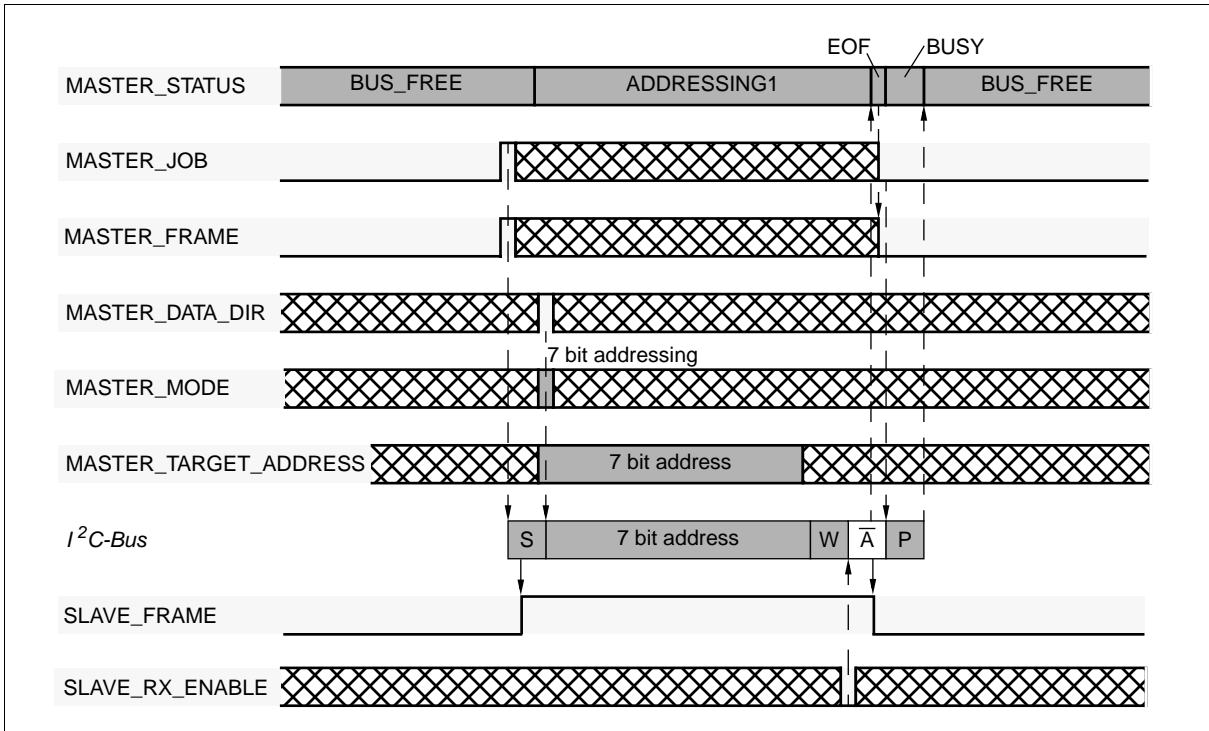


Figure 16: Delay Due to Late MASTER_DATAIN_VAL

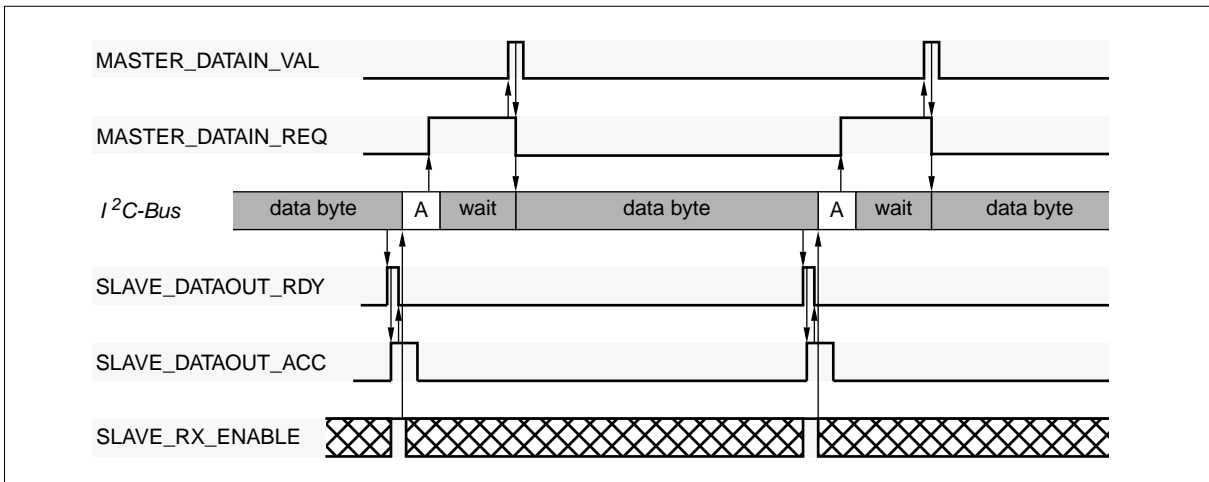


Figure 17: Delay Due to Late SLAVE_DATAOUT_ACC

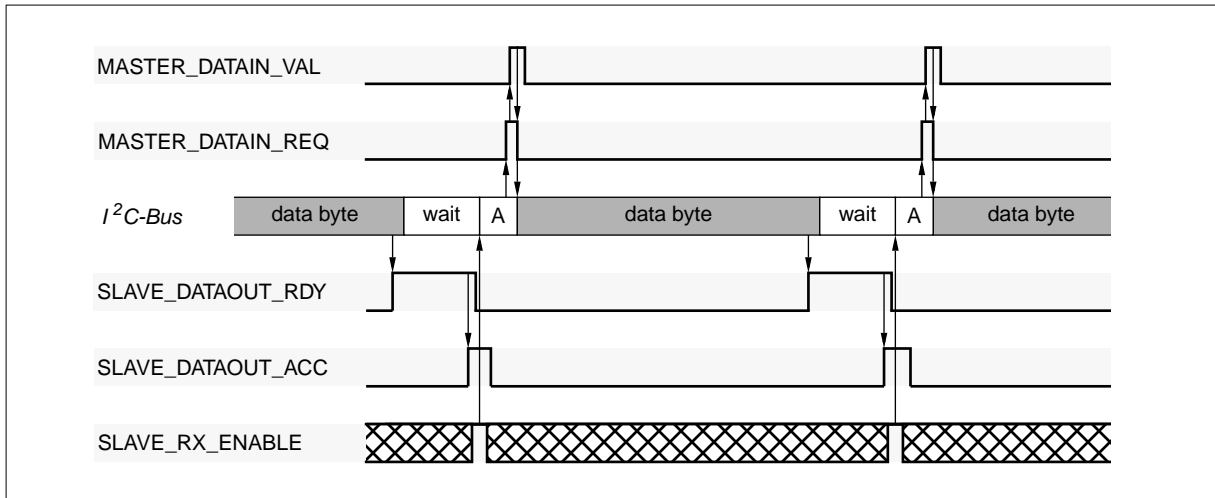


Figure 18: Eventual Handshaking Speed-Up Due to Premature DATAIN_VAL and/or DATAOUT_ACC

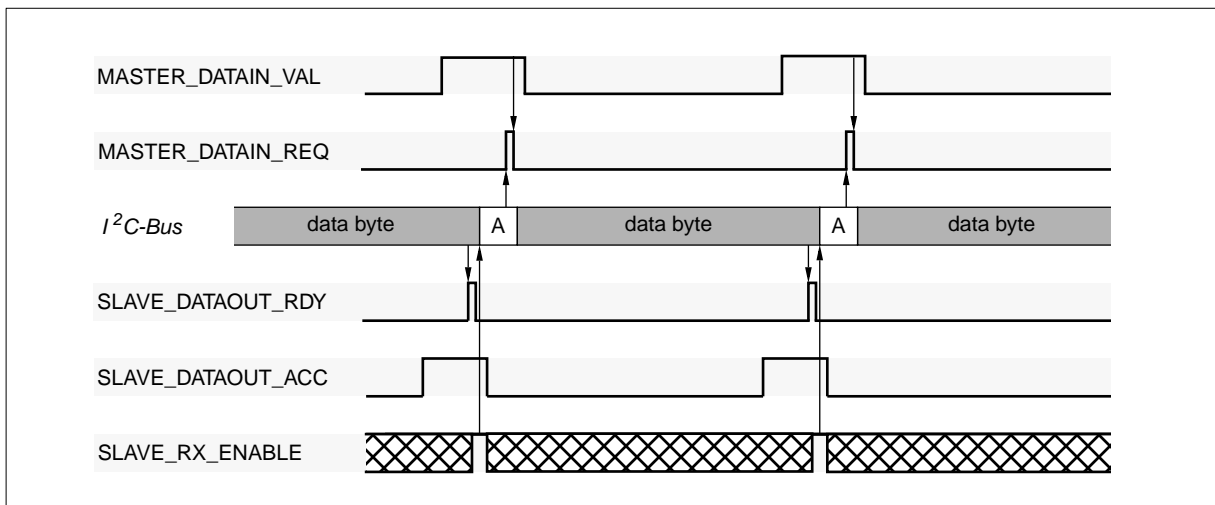
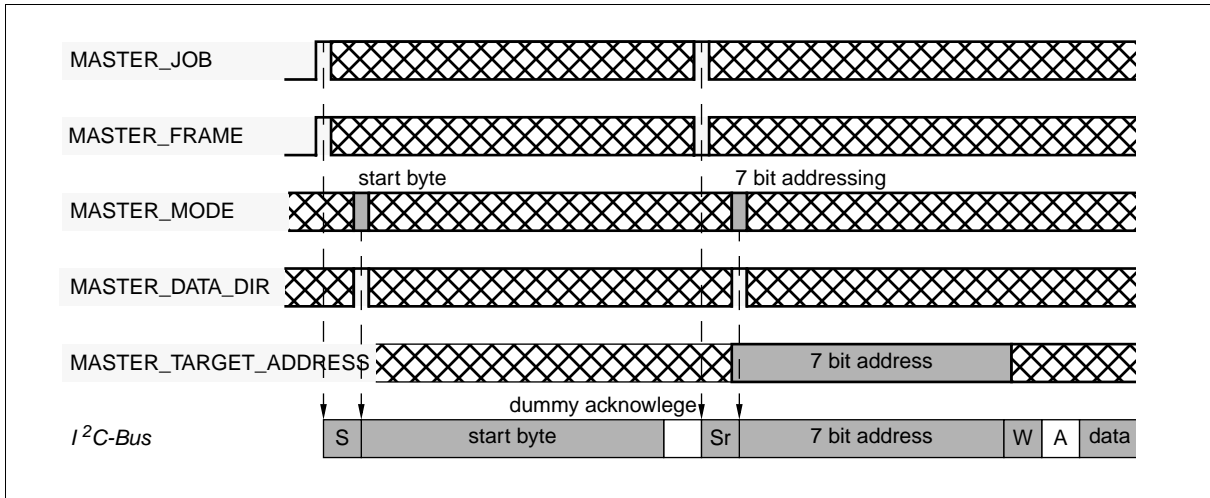


Figure 19: Start Byte



Application Notes

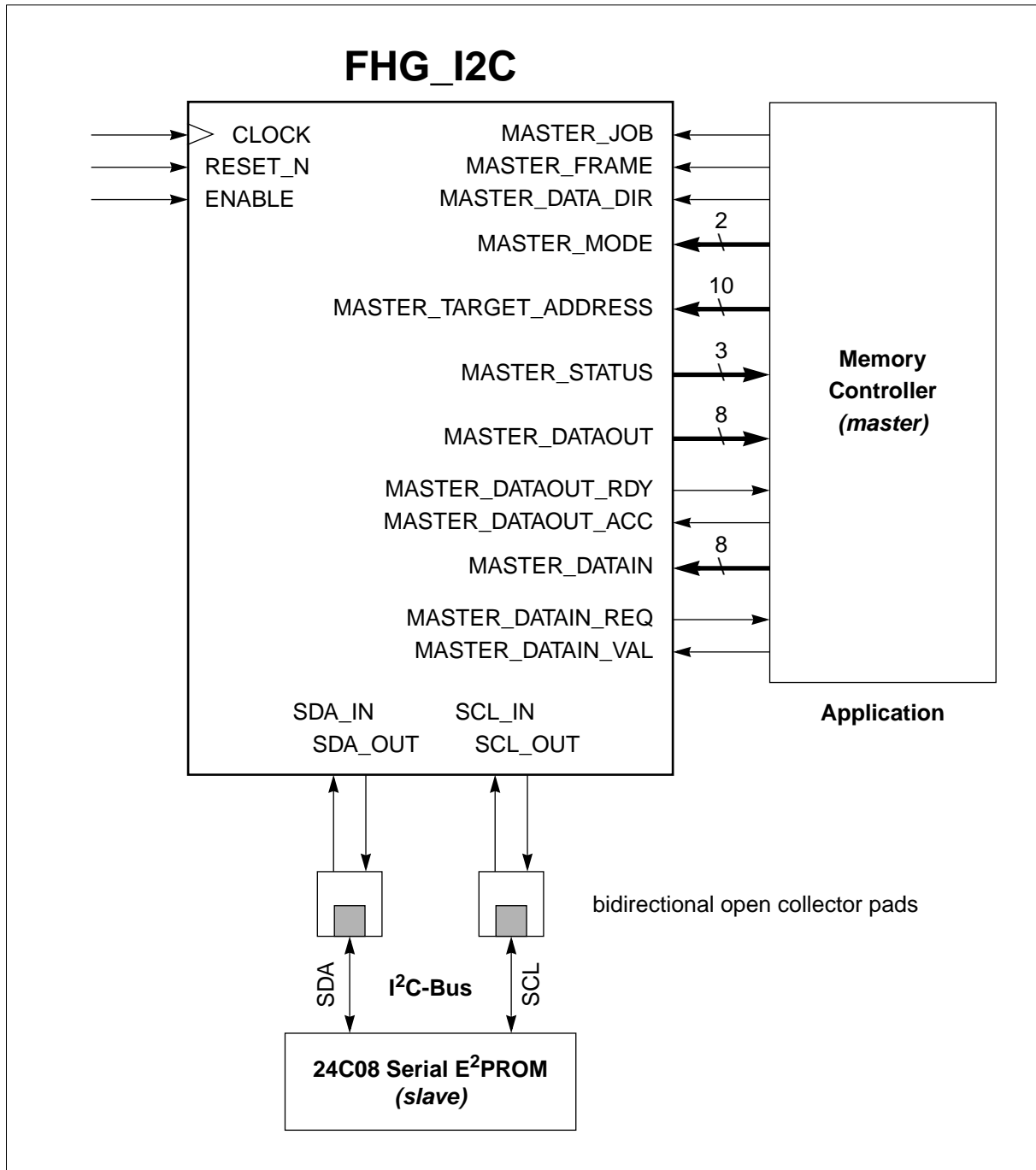
Master Application

The following figure shows a typical master-application, with *setup-parameters* being stored in an external non-volatile memory and load on system initialization.

The I²C-bus interface is implemented as a master-receiver/transmitter with 7 bit addressing and a transfer rate of 100 kbit/s. Being a VHDL-component, all remaining connections must be set

either with constants (inputs) or with the VHDL-keyword *open* (outputs).

Figure 20: Master Application



Slave Application

This application is a common FIFO expanded for I²C-bus operation. The target address is 10 bits wide and has the arbitrary value 0x2A (decimal 42).

The I²C-bus interface is implemented as a slave-receiver/transmitter with 7/10 bit addressing and a transfer rate of 400 kbit/s (fast mode 9:16). Being a VHDL-component, all remaining connections must be set either with constants (inputs) or with the VHDL-keyword *open* (outputs).

Figure 21: Slave Application

