OpenCores

www.opencores.org

I^2C Multiple Bus Controller

IP Core Specification

Author: Sergey Shuvalkin
sshuv2@opencores.org

Revision 1.0a
May 4, 2016

WISHBONE COMPATIBLE

Document Revision History

Revision	Date	Author	Description
1.0	04/29/2016	Sergey Shuvalkin	Initial release
1.0a	05/04/2016	Sergey Shuvalkin	Minor improvements

Reference Documents

1. UM10204, I2C-bus specification and user manual, Rev. 6 – April 2014.
2. MNL-AVABUSREF, Avalon Interface Specifications, Ver. 14.1 – March 2015.
3. Wishbone B4, – 2010.

Table of Contents

1 Introduction

This specification defines the architecture, hardware interface and parameterization options for the I²C Multiple Bus Controller (IICMB) core.

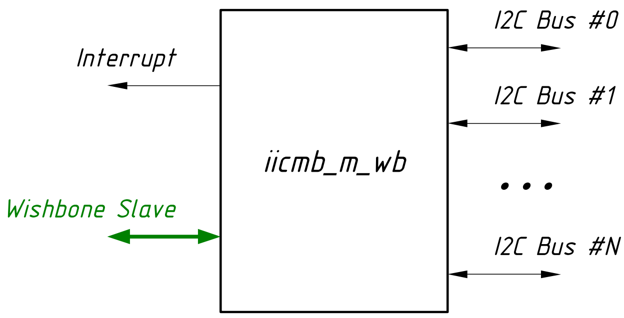

Figure 1: High-level view on the interfaces

The IICMB core provides a low-speed, two-wire, bidirectional serial bus interfaces compliant to industry standard I²C protocol. The key feature of the core is its ability to control several connected I²C buses effectively reducing complexity of system.

At any given moment the IICMB core works with a single I²C bus chosen from the range of connected buses (in this document such bus is called *selected bus*). When work with a particular selected bus is finished, user can switch to another one to continue configuration of other peripherals. Every connected I2C bus is recognized by its number, or *bus ID*.

Note: The current version of the core supports master only functionality. Slave mode is under development.

Features:

- Compatible with Philips I²C standard
- Works with up to 16 distinct I²C buses
- Statically configurable system bus clock frequency
- Statically configurable desired clock frequencies of I²C buses
- Multi-master clock synchronization
- Multi-master arbitration
- Clock stretching
- Digital filtering of SCL and SDA inputs
- Standard (up to 100 kHz) and Fast (up to 400 kHz) mode operation
- Connects as 8-bit slave on Wishbone bus
- Connects as 32-bit slave on Avalon-MM bus

2 Architecture

The core is provided with three examples of its top level (`iicmb_m_wb.vhd`, `iicmb_m_av.vhd` and `iic_m_sq.vhd`). Two of them are designed for Wishbone and Avalon-MM buses, while third version is a sequencer based one for deeply embedded applications without any system bus at all.

In the center of the IICMB core is `iicmb_m.vhd` module which integrates byte- and bit-level master mode FSMs together with I²C bus multiplexer functionality. It is controlled with byte-level commands sent through the so-called *Generic Interface*.

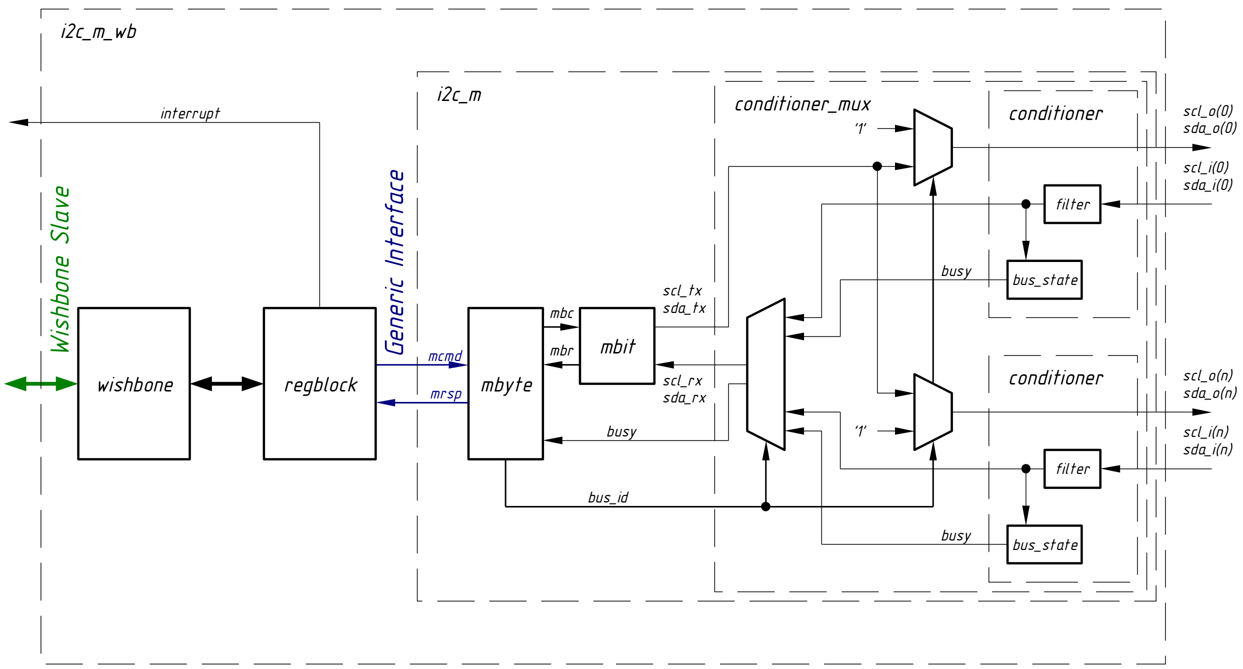

Figure 2: Block diagram of the Wishbone version of the top level

The `wishbone.vhd` module connects Wishbone bus to register block (`regblock.vhd`), which converts system bus accesses to byte-level commands of the Generic Interface.

SCL and SDA inputs are digitally filtered to suppress unwanted spikes and to cope with long rising time of the I²C bus signals. The `bus_state.vhd` modules independently monitor busy states of all connected buses.

The `conditioner_mux.vhd` module, controlled by `bus_id` input, performs switching between connected I²C buses.

The `mbit.vhd` and `mbyte.vhd` implement bit-level and byte-level FSMs, generating appropriate SCL and SDA waveforms in accordance to I²C Bus Specification.

3 Operation

3.1 Byte-level FSM

The byte-level FSM module (`mbyte.vhd`) communicates with upper level through so called Generic Interface. It accepts several byte-level commands, listed in the Table 1 below. After completion, each command is answered with an appropriate response. The main responsibility of the `mbyte.vhd` module is to translate byte-level commands to one or more commands for bit-level FSM (`mbit.vhd`).

Reception of a response is a mark of completion of the previously issued command. It is an error to send next command before previous command is responded. Such a command is ignored.

Command	Code	Parameter	Description
Start	“100”	–	If bus is not captured yet: issue Start Condition and capture selected bus. If bus captured: issue Repeated Start Condition.
Stop	“101”	–	Issue Stop Condition and free selected bus.
Read With Ack	“010”	–	Receive a byte with acknowledge.
Read With Nak	“011”	–	Receive a byte with not-acknowledge.
Write	“001”	Byte of data	Transmit the byte given as a parameter.
Set Bus	“110”	Bus number (ID)	Connect to the specified bus (select bus).
Wait	“000”	Milliseconds	Do nothing for specified amount of time.

Table 1: Byte-level commands

Response	Code	Parameter	Description
Done	“000”	–	Command completed.
Arbitration Lost	“010”	–	Arbitration lost. Selected bus is freed, FSMs are set to their idle states.
No Acknowledge	“001”	–	Byte written got no acknowledge.
Byte	“100”	Byte of data	Byte of data received.
Error	“011”	–	Something went wrong.

Table 2: Byte-level responses

	Done	Arb. Lost	No Ack.	Byte	Error
Start	+	+			
Stop	+				
Read With Ack				+	+
Read With Nak		+		+	+
Write	+	+	+		+
Set Bus	+				+
Wait	+				+

Table 3: Possible responses to byte-level commands

The following diagram depicts the byte-level FSM:

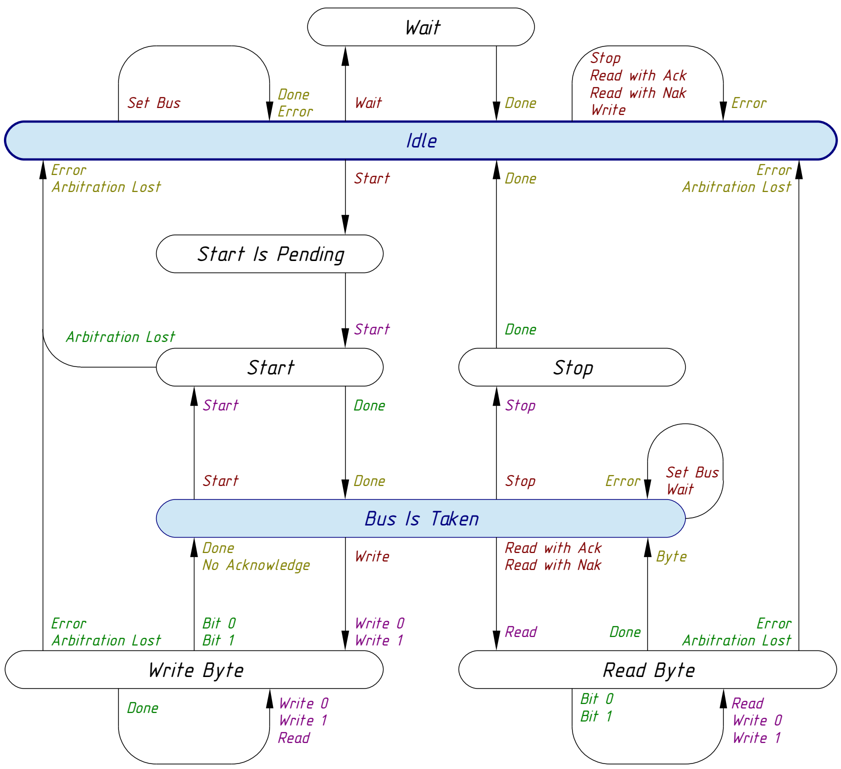

Figure 3: Byte-level FSM

3.2 Generic Interface

The Generic Interface consists of the following signals:

Signal Name	I/O	Description
mcmd_wr	Input	Byte-level command write (active high).
mcmd_id[2:0]	Input	Byte-level command ID.
mcmd_data[7:0]	Input	Byte-level command parameter.
mrsp_wr	Output	Byte-level response write (active high).
mrsp_id[2:0]	Output	Byte-level response ID.
mrsp_data[7:0]	Output	Byte-level response parameter.

Table 4: Generic Interface signals

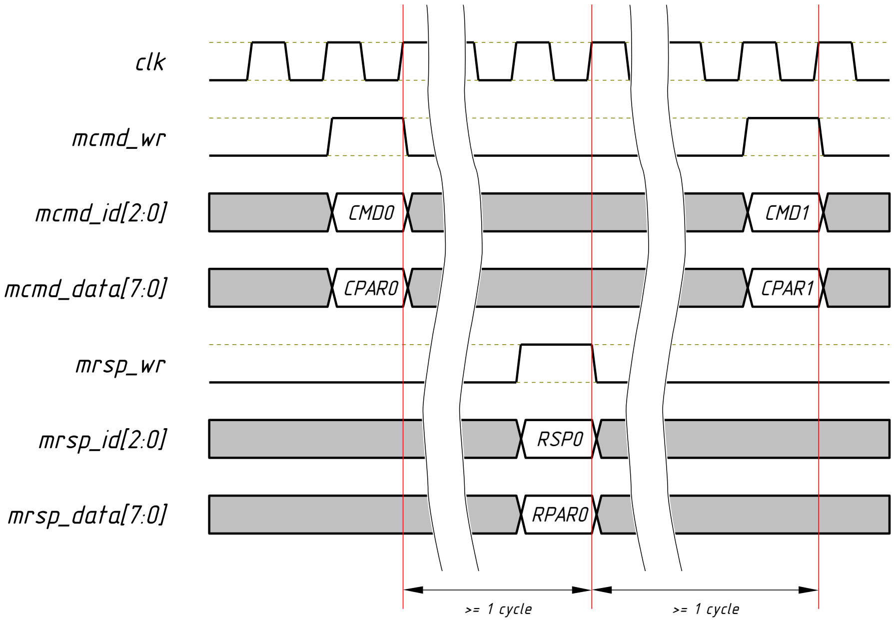

Figure 4: Generic interface timing diagram

3.3 Bit-level FSM

Bit-level commands and responses are hidden from the user of the core, but listed here for better understanding of how the two FSMs interact with each other.

Bit-level commands and responses have no parameters.

Command	Description
Start	Issue Start Condition or Repeated Start Condition.
Stop	Issue Stop Condition.
Write 0	Send bit '0'.
Write 1	Send bit '1'.
Read	Receive a bit.

Table 5: Bit-level commands

Response	Description
Done	Command completed.
Arbitration Lost	Arbitration lost.
Bit 0	Bit '0' received.
Bit 1	Bit '1' received.
Error	Something gone wrong.

Table 6: Bit-level responses

	Done	Arb. Lost	Bit 0	Bit 1	Error
Start	+	+			
Stop	+				
Write 0	+				+
Write 1	+	+			+
Read			+	+	+

Table 7: Possible responses to bit-level commands

The following diagram depicts the bit-level FSM:

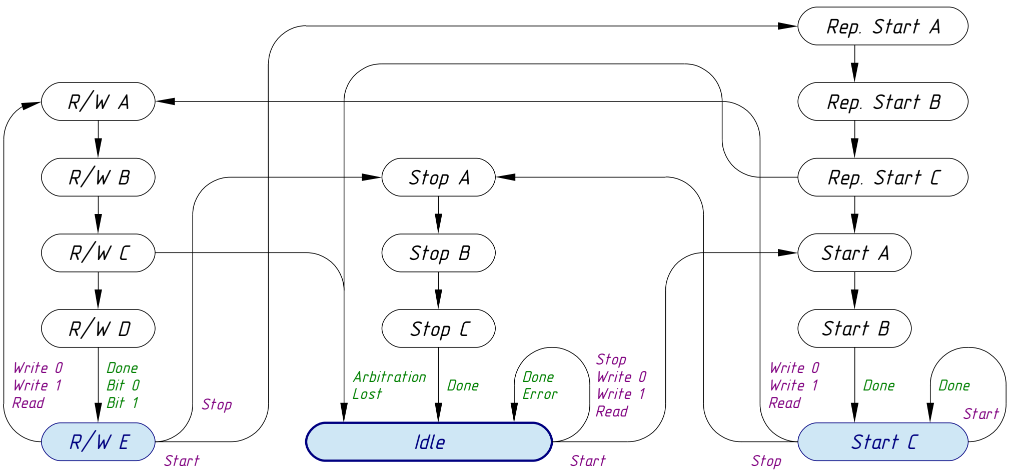

Figure 5: Bit-level FSM

Traversing along bit-level FSM states produces the following waveforms on SCL and SDA bus signals.

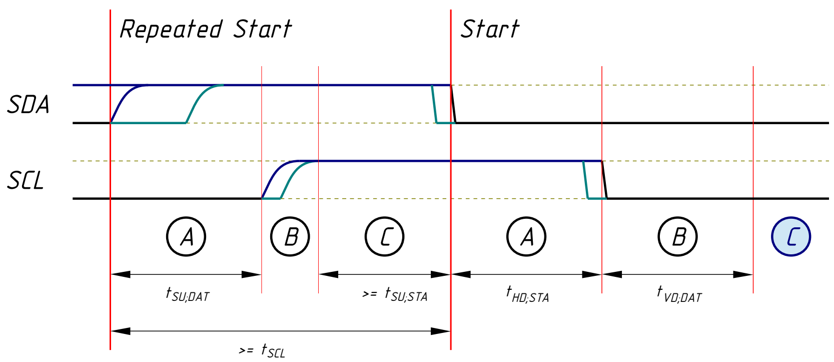

Figure 6: Start and Repeated Start conditions waveform

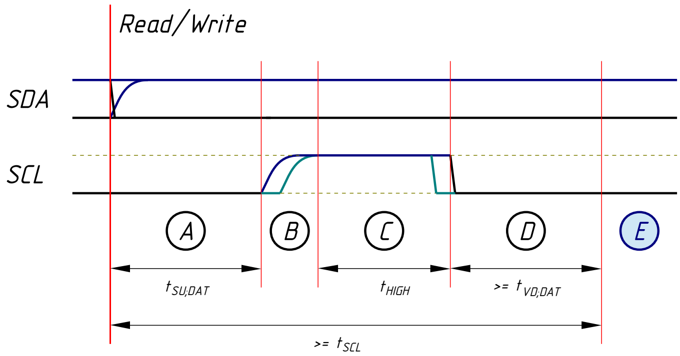

Figure 7: Read/Write waveform

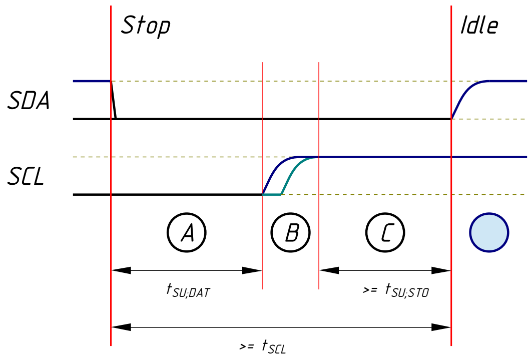

Figure 8: Stop Condition waveform

4 Interfaces

4.1 Parameters

These are generics on the top level VHDL entities of the core.

Name	Type	Default Value	Description
g_bus_num	positive	1	Number of connected I²C buses
g_f_clk	real	100000.0	Frequency of system clock (in kHz)
g_f_scl_0	real	100.0	Frequency of SCL clock of I²C bus #0 (in kHz)
g_f_scl_1	real	100.0	Frequency of SCL clock of I²C bus #1 (in kHz)
g_f_scl_2	real	100.0	Frequency of SCL clock of I²C bus #2 (in kHz)
g_f_scl_3	real	100.0	Frequency of SCL clock of I²C bus #3 (in kHz)
g_f_scl_4	real	100.0	Frequency of SCL clock of I²C bus #4 (in kHz)
g_f_scl_5	real	100.0	Frequency of SCL clock of I²C bus #5 (in kHz)
g_f_scl_6	real	100.0	Frequency of SCL clock of I²C bus #6 (in kHz)
g_f_scl_7	real	100.0	Frequency of SCL clock of I²C bus #7 (in kHz)
g_f_scl_8	real	100.0	Frequency of SCL clock of I²C bus #8 (in kHz)
g_f_scl_9	real	100.0	Frequency of SCL clock of I²C bus #9 (in kHz)
g_f_scl_a	real	100.0	Frequency of SCL clock of I²C bus #10 (in kHz)
g_f_scl_b	real	100.0	Frequency of SCL clock of I²C bus #11 (in kHz)
g_f_scl_c	real	100.0	Frequency of SCL clock of I²C bus #12 (in kHz)
g_f_scl_d	real	100.0	Frequency of SCL clock of I²C bus #13 (in kHz)
g_f_scl_e	real	100.0	Frequency of SCL clock of I²C bus #14 (in kHz)
g_f_scl_f	real	100.0	Frequency of SCL clock of I²C bus #15 (in kHz)

Table 8: Core parameters

Allowed range of the `g_bus_num` is from 1 to 16.

The `g_f_clk` parameter specifies the main clock frequency. Depending on top level version the main clock input is `clk_i` or `clk`.

Actual frequencies of I²C clocks will be little bit lower than specified with parameters `g_f_scl_N`.

4.2 Wishbone Interface Signals

Wishbone version of the core top level has the following signals:

Signal Name	I/O	Description
clk_i	Input	Wishbone clock. Main clock for the controller.
rst_i	Input	Synchronous reset. Active high.
cyc_i	Input	Valid bus cycle.
stb_i	Input	Strobe signal/Core select.
ack_o	Output	Bus cycle acknowledge.
adr_i[1:0]	Input	Lower address bits.
we_i	Input	Write enable.
dat_i[7:0]	Input	Data input.
dat_o[7:0]	Output	Data output.

Table 9: Wishbone interface signals

The frequency of `clk_i` clock (in kHz) should be indicated in `g_f_clk` parameter.

For more information about Wishbone signals please refer to Wishbone B4 Specification.

4.3 Avalon-MM Interface Signals

Avalon-MM version of the core top level has the following signals:

Signal Name	I/O	Description
clk	Input	Avalon-MM clock. Main clock for the controller.
s_rst	Input	Synchronous reset. Active high.
waitrequest	Output	Wait request.
readdata[31:0]	Output	Data output.
readdatavalid	Output	Data validity indication.
writedata[31:0]	Input	Data input.
write	Input	Indicates a write transfer.
read	Input	Indicates a read transfer.
byteenable[3:0]	Input	Enables specific byte lane(s) during transfer

Table 10: Avalon-MM interface Signals

There is no address signal in this group. This is because we have only 4 bytes of registers and all of them can be accessed in one single read or write.

The frequency of clk clock (in kHz) should be indicated in g_f_clk parameter.

For more information about Avalon-MM signals please refer to Avalon Interface Specifications.

4.4 Sequencer Top Level Signals

The iicmb_sq_m.vhd is example of the top level without using any system bus interface. It includes a simple sequencer, which allows to play back any sequence of byte-level commands on the Generic Interface. The sequencer is controlled by the following signals:

Signal Name	I/O	Description
clk	Input	Main clock for the controller.
s_rst	Input	Synchronous reset. Active high.
cs_start	Input	Start executing the command sequence.
cs_busy	Output	Sequencer busy status. '1' = command sequence is being executed; '0' = command sequence finished executing.
cs_status[2:0]	Output	Command sequence execution status.

Table 11: Sequencer related interface signals

The frequency of clk clock (in kHz) should be indicated in g_f_clk parameter.

The cs_status signal takes the codes of byte-level responses: Done, Arbitration Lost, No Acknowledge and Error.

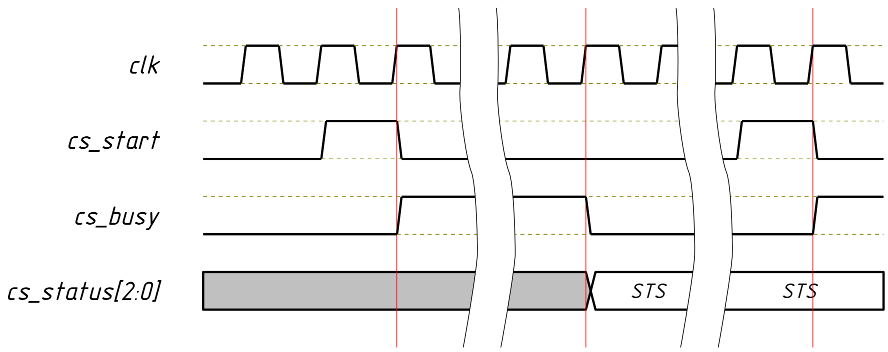

Figure 9: Sequencer control timing diagram

Issuing another cs_start pulse after cs_busy has returned back to '0' repeats command sequence again.

The iicmb_sq_m.vhd module has additional parameter (VHDL generic), g_cmd, which holds the sequence of byte-level commands to playback:

Name	Type	Default Value	Description
g_cmd[– : –]	seq_cmd_type_array	c_empty_array	Sequence of byte-level commands in special format.

Table 12: Additional parameter of sequencer-based top-level module.

The `seq_cmd_type_array` type and `c_empty_array` constant are defined in `iicmb_pkg.vhd` VHDL package. The same package also defines several functions which simplify defining byte-level command sequences. Example of such sequence can be found in `iicmb_sq_m_tb.vhd` testbench file.

4.5 I²C Serial Lines

Signal Name	I/O	Description
`scl_i[0:g_bus_num – 1]`	Input	Serial clock line inputs.
`sda_i[0:g_bus_num – 1]`	Input	Serial data line inputs.
`scl_o[0:g_bus_num – 1]`	Output	Serial clock line outputs.
`sda_o[0:g_bus_num – 1]`	Output	Serial data line outputs.

Table 13: I²C serial lines

Physical layer of I²C serial lines must be created outside of the IICMB core by means of tri-state buffers which are connected in the following way:

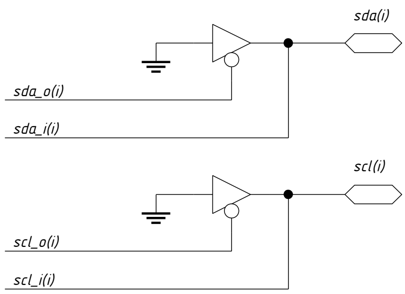

Figure 10: Connecting tri-state buffers

Such buffers can be instantiated implicitly, for example, with the VHDL code below:

```
phy_gen:
for i in 0 to g_bus_num - 1 generate
  scl_i(i) <= scl(i);
```

```
  scl(i)   <= '0' when (scl_o(i) = '0') else 'Z';
  sda_i(i) <= sda(i);
  sda(i)   <= '0' when (sda_o(i) = '0') else 'Z';
end generate phy_gen;
```

4.6 Other Signals

Signal Name	I/O	Description
irq	Output	Interrupt request.

Table 14: Other signals

The irq signal is a level sensitive interrupt (active level = '1'). The interrupt request is generated when a byte-level command has been completed and Interrupt Enable bit (IE) in the Control/Status Register (CSR) is equal to '1'.

It is generated by register block and can be cleared (reset to '0') by reading CMDR register (see register descriptions below).

5 Registers

Wishbone and Avalon-MM based top level modules of IICMB controller (`iicmb_m_wb.vhd` and `iicmb_m_av.vhd`) contain a register block with the following registers. Note, that the sequencer based top level (`iicmb_m_sq.vhd`) does without them.

Name	Offset	Access	Description
CSR	`0x00`	R/W	Control/Status Register
DPR	`0x01`	R/W	Data/Parameter Register
CMDR	`0x02`	R/W	Command Register
FSMR	`0x03`	RO	FSM States Register

Table 15: IICMB Register list

5.1 Control/Status Register (CSR)

Control/Status register, offset 0x00:

	7	6	5	4	3	2	1	0
`0x00`	E	IE	BB	BC	Bus ID			
	R/W	R/W	RO	RO	RO			
	'0'	'0'	'0'	'0'	"0000"			

Name	Bits	Access	Reset Value	Description
E	7	R/W	'0'	Enable. Effectively resets IICMB core. '0' = IICMB is disabled; '1' = IICMB is enabled.
IE	6	R/W	'0'	Interrupt Enable. '0' = `irq` output is disabled; '1' = `irq` output is enabled.
BB	5	RO	'0'	Bus Busy. Indicates selected bus state. '0' = Selected bus is idle; '1' = Selected bus is busy.
BC	4	RO	'0'	Bus Captured. Indicates when IICMB has captured the selected bus. '0' = Selected bus isn't captured by IICMB. '1' = Selected bus is captured by IICMB.
Bus ID	3 – 0	RO	"0000"	Bus ID. Indicates selected bus ID.

After reset or power-up the Enable bit (E) has value '0'. This effectively resets the IICMB core and prohibits writing to other registers (except bits E and IE of the CSR). Therefore, the first action after reset should be writing to CSR register to set bit E to '1' and, by the way, to set bit IE to any desired value.

Bus Bysy bit (BB) and Bus Captured (BC) bit (as well as Bus ID field) should be used mostly for diagnostic purposes. For example, there is no need to refer to BB bit to determine bus busyness before giving a Start command – this is done automatically by the byte-level FSM.

5.2 Data/Parameter Register (DPR)

Data/Parameter register, offset 0x01:

	7	6	5	4	3	2	1	0
`0x01`	Data							
	R/W							
	"00000000"							

Name	Bits	Access	Reset Value	Description
Data	7 – 0	R/W	"00000000"	Data or a parameter for a byte-level command.

Before issuing a Write command, a byte to transmit has to be written into the DPR.

Reading DPR register returns last byte received via I²C bus.

Parameters for the Set Bus and Wait commands are also written into the DPR. These two commands interpret content of the DPR register as unsigned 8-bit integer. The parameter written for the Set Bus command should be in range from 0 to `g_num_bus` – 1, otherwise the command returns ERR status. The parameter for the Wait command is number of *milliseconds* to wait.

5.3 Command Register (CMDR)

Command register, offset 0x02:

	7	6	5	4	3	2	1	0
`0x02`	DON	NAK	AL	ERR	R	CMD		
	RO	RO	RO	RO	RO	R/W		
	'1'	'0'	'0'	'0'	'0'	"000"		

Name	Bits	Access	Reset Value	Description
DON	7	R/W	'1'	Done. Indicates command completion.

Name	Bits	Access	Reset Value	Description
				'0' = FSMs are busy, or a command was completed with one of the three other completion statuses; '1' = Command completed normally.
NAK	6	R/W	'0'	Data write was not acknowledged. '0' = FSMs are busy, or a command was completed with one of the three other completion statuses; '1' = Write command was not acknowledged.
AL	5	RO	'0'	Arbitration Lost. '0' = FSMs are busy, or a command was completed with one of the three other completion statuses; '1' = Arbitration was lost during command execution.
ERR	4	RO	'0'	Error indication. '0' = FSMs are busy, or a command was completed with one of the three other completion statuses. '1' = Last command terminated with an error.
R	3	RO	'0'	Reserved bit.
CMD	2 – 0	R/W	“000”	Byte-level command code. Write to this field starts execution of the correspondent command. Reading returns last written command code.

Writing to the register starts execution of a byte-level command and clears all status bits (DON, NAK, AL and ERR).

During command execution all status bits (DON, NAK, AL and ERR) are '0'.

When a command is completed, one of the four status bits (DON, NAK, AL or ERR) becomes '1', depending on the completion results. In the same moment, the interrupt output is activated (`irq` = '1'), if it is enabled by setting bit IE in CSR to level '1'.

Reading the register clears the interrupt output (`irq` = '0'), while the status bits retain their values.

5.4 FSM States Register

FSM States register, offset 0x03:

	7	6	5	4	3	2	1	0
0x03	BYTE_FSM				BIT_FSM			
	RO				RO			
	“0000”				“0000”			

Name	Bits	Access	Reset Value	Description
BYTE_FSM	7 – 4	RO	“0000”	Current state of Byte-level FSM.
BIT_FSM	3 – 0	RO	“0000”	Current state of Bit-level FSM.

The register displays current states of bit- and byte-level FSMs. Knowing these states is not needed for normal operation, but might be useful when analyzing erroneous situations.

6 Programming Examples

6.1 Example 1

Task: Enable the IICMB core after power-up.

System bus actions:

1. Write byte “1xxxxxxx” to the CSR register. This sets bit E to '1', enabling the core.

6.2 Example 2

Task: Disable or reset the IICMB core.

System bus actions:

1. Write byte “0xxxxxxx” to the CSR register. This sets bit E to '0', disabling the core.

6.3 Example 3

Task: Write a byte 0x78 to a slave with address 0x22, residing on I^{2}C bus #5.

System bus actions:

1. Write byte 0x05 to the DPR. This is the ID of desired I^{2}C bus.
2. Write byte “xxxxx110” to the CMDR. This is Set Bus command.
3. Wait for interrupt or until DON bit of CMDR reads '1'.
4. Write byte “xxxxx100” to the CMDR. This is Start command.
5. Wait for interrupt or until DON bit of CMDR reads '1'.
6. Write byte 0x44 to the DPR. This is the slave address 0x22 shifted 1 bit to the left + rightmost bit = '0', which means writing.
7. Write byte “xxxxx001” to the CMDR. This is Write command.
8. Wait for interrupt or until DON bit of CMDR reads '1'. If instead of DON the NAK bit is '1', then slave doesn't respond.
9. Write byte 0x78 to the DPR. This is the byte to be written.
10. Write byte “xxxxx001” to the CMDR. This is Write command.
11. Wait for interrupt or until DON bit of CMDR reads '1'.
12. Write byte “xxxxx101” to the CMDR. This is Stop command.
13. Wait for interrupt or until DON bit of CMDR reads '1'.

6.4 Example 4

Task: Write a byte of data to an I^{2}C memory device. Slave address = 0x23. Slave resides on I^{2}C bus #4. Memory location to write to = 0x9B. Byte of data to write = 0xEE.

System bus actions:

1. Write byte 0x04 to the DPR. This is the ID of desired I^{2}C bus.
2. Write byte “xxxxx110” to the CMDR. This is Set Bus command.
3. Wait for interrupt or until DON bit of CMDR reads '1'.
4. Write byte “xxxxx100” to the CMDR. This is Start command.
5. Wait for interrupt or until DON bit of CMDR reads '1'.
6. Write byte 0x46 to the DPR. This is the slave address 0x23 shifted 1 bit to the left + rightmost bit is '0' which means writing.
7. Write byte “xxxxx001” to the CMDR. This is Write command.
8. Wait for interrupt or until DON bit of CMDR reads '1'. If instead of DON the NAK bit is '1', then slave doesn't respond.
9. Write byte 0x9B to the DPR. This is the memory location to write to.
10. Write byte “xxxxx001” to the CMDR. This is Write command.
11. Wait for interrupt or until DON bit of CMDR reads '1'.
12. Write byte 0xEE to the DPR. This is the byte of data to write.
13. Write byte “xxxxx001” to the CMDR. This is Write command.
14. Wait for interrupt or until DON bit of CMDR reads '1'.
15. Write byte “xxxxx101” to the CMDR. This is Stop command.
16. Wait for interrupt or until DON bit of CMDR reads '1'.

6.5 Example 5

Task: Read a byte of data from an I^{2}C memory device. Slave address = 0x44. Slave resides on I^{2}C bus #1. Memory location to read from = 0xAA.

System bus actions:

1. Write byte 0x01 to the DPR. This is the ID of desired I^{2}C bus.
2. Write byte “xxxxx110” to the CMDR. This is Set Bus command.
3. Wait for interrupt or until DON bit of CMDR reads '1'.
4. Write byte “xxxxx100” to the CMDR. This is Start command.
5. Wait for interrupt or until DON bit of CMDR reads '1'.
6. Write byte 0x88 to the DPR. This is the slave address 0x44 shifted 1 bit to the left +

rightmost bit is '0' which means writing.

7. Write byte “xxxxx001” to the CMDR. This is Write command.
8. Wait for interrupt or until DON bit of CMDR reads '1'. If instead of DON the NAK bit is '1', then slave doesn't respond.
9. Write byte 0xAA to the DPR. This is the memory location to read from.
10. Write byte “xxxxx001” to the CMDR. This is Write command.
11. Wait for interrupt or until DON bit of CMDR reads '1'.
12. Write byte “xxxxx100” to the CMDR. This is Start command (repeated start).
13. Wait for interrupt or until DON bit of CMDR reads '1'.
14. Write byte 0x89 to the DPR. This is the slave address 0x44 shifted 1 bit to the left + rightmost bit is '1' which means reading.
15. Write byte “xxxxx001” to the CMDR. This is Write command.
16. Wait for interrupt or until DON bit of CMDR reads '1'.
17. Write byte “xxxxx011” to the CMDR. This is Read With Nak command.
18. Wait for interrupt or until DON bit of CMDR reads '1'.
19. Read DPR to get received byte of data.
20. Write byte “xxxxx101” to the CMDR. This is Stop command.
21. Wait for interrupt or until DON bit of CMDR reads '1'.

7 Project Directory Structure

```
iicmb                                  Project directory
├ doc                                  Documentation directory
│  ├ src                               Specification source files
│  │  └ iicmb_spec.odt
│  └ iicmb_spec.pdf                    Specification (this document)
├ src                                  VHDL sources of the core
│  ├ avalon_mm.vhd
│  ├ bus_state.vhd
│  ├ conditioner.vhd
│  ├ conditioner_mux.vhd
│  ├ filter.vhd
│  ├ iicmb_int_pkg.vhd                 Package for internal definitions
│  ├ iicmb_m.vhd
│  ├ iicmb_m_am.vhd                    Top level (Avalon-MM version)
│  ├ iicmb_m_sq.vhd                    Top level (Sequencer version)
│  ├ iicmb_m_wb.vhd                    Top level (Wishbone version)
│  ├ iicmb_pkg.vhd                     Global package
│  ├ mbit.vhd
│  ├ mbyte.vhd
│  ├ regblock.vhd
│  ├ sequencer.vhd
│  └ wishbone.vhd
├ src_tb                               VHDL sources of the test benches
│  ├ iicmb_m_sq_arb_tb.vhd             Testbench for the sequencer-based top level
│  ├ iicmb_m_sq_tb.vhd                 Testbench for the sequencer-based top level
│  ├ iicmb_m_tb.vhd                    Testbench for the iicmb_m.vhd module
│  └ iicmb_m_wb.vhd                    Testbench for the Wishbone-based top level
└ sim                                  Simulation directory
```

8 Hierarchy Of Modules

Hierarchy of modules with Wishbone top level:

- iicmb_pkg.vhd
- iicmb_int_pkg.vhd
- iicmb_m_wb.vhd
  - wishbone.vhd
  - regblock.vhd
  - iicmb_m.vhd
    - mbyte.vhd
    - mbit.vhd
    - conditioner_mux.vhd
      - conditioner.vhd
        - filter.vhd
        - bus_state.vhd

Hierarchy of modules with Avalon-MM top level:

- iicmb_pkg.vhd
- iicmb_int_pkg.vhd
- iicmb_m_av.vhd
  - avalon_mm.vhd
  - regblock.vhd
  - iicmb_m.vhd
    - mbyte.vhd
    - mbit.vhd
    - conditioner_mux.vhd
      - conditioner.vhd
        - filter.vhd
        - bus_state.vhd

Hierarchy of modules with sequencer top level:

iicmb_pkg.vhd
iicmb_int_pkg.vhd
iicmb_m_sq.vhd
- sequencer.vhd
- iicmb_m.vhd
  - mbyte.vhd
  - mbit.vhd
  - conditioner_mux.vhd
    - conditioner.vhd
      - filter.vhd
      - bus_state.vhd

9 Implementation Results

9.1 Setup 1

Top level module: iicmb_m_wb.vhd.

Number of I²C buses (`g_num_bus`) = 1.

System clock frequency (`g_f_clk`) = 100 MHz.

I²C bus clock frequency (`g_f_scl_0`) = 100 kHz.

Manufacturer	Family	Device	F_{max}	Registers	Logic
Altera	Cyclone IV E	EP4CE6E22C9L	130 MHz	172	463 LEs
	Cyclone V GX	5CGXBC3B7F23C8	183 MHz	233	269 ALMs
	MAX II	EPM2210F324I5	85 MHz	172	465 LEs
	Arria II GX	EP2AGX45CU17C6	260 MHz	173	413 ALUTs
	Arria V GZ	5AGZME1E3H29C4	455 MHz	236	269 ALMs
	Stratix IV	EP4SGX70DF29C4	350 MHz	173	409 ALUTs
	Stratix V	5SGSMD3E3H29C4	434 MHz	230	269 ALMs
	MAX 10	10M02DCU324C8G	173 MHz	172	462 LEs
	Arria 10	10AS016C4U19E3LG	510 MHz	230	270 ALMs
Xilinx	Spartan-3A	xc3s50a-5tq144	141 MHz	167	414 LUTs
	Spartan-6	xc6slx4-3tqg144	216 MHz	154	271 LUTs
	Virtex-4	xc4vfx12-12-sf363	266 MHz	164	430 LUTs
	Virtex-5	xc5vlx20t-2-ff323	289 MHz	164	344 LUTs
	Virtex-6	xc6vcx75t-2-ff484	325 MHz	154	264 LUTs
	Kintex-7	xc7k70t-3-fbg676	400 MHz	154	309 LUTs
	Artix-7	xc7a100t-3-csg324	322 MHz	154	268 LUTs
	Virtex-7	xc7vx330t-3-ffg1157	426 MHz	154	309 LUTs

9.2 Setup 2

Top level module: iicmb_m_wb.vhd.

Number of I^{2}C buses (`g_num_bus`) = 16.

System clock frequency (`g_f_clk`) = 100 MHz.

I^{2}C bus clock frequencies:

`g_f_scl_0` = 100 kHz, `g_f_scl_1` = 120 kHz, `g_f_scl_2` = 130 kHz, `g_f_scl_3` = 200 kHz, `g_f_scl_4` = 50 kHz, others = 30 kHz.

Manufacturer	Family	Device	F_{max}	Registers	Logic
Altera	Cyclone IV E	EP4CE6E22C9L	93 MHz	667	1520 LEs
	Cyclone V GX	5CGXBC3B7F23C8	152 MHz	853	785 ALMs
	MAX II	EPM2210F324I5	68 MHz	667	1496 LEs
	Arria II GX	EP2AGX45CU17C6	234 MHz	677	1238 ALUTs
	Arria V GZ	5AGZME1E3H29C4	330 MHz	803	784 ALMs
	Stratix IV	EP4SGX70DF29C4	290 MHz	667	1234 ALUTs
	Stratix V	5SGSMD3E3H29C4	368 MHz	794	784 ALMs
	MAX 10	10M02DCU324C8G	148 MHz	667	1516 LEs
	Arria 10	10AS016C4U19E3LG	400 MHz	804	785 ALMs
Xilinx	Spartan-3A	xc3s50a-5tq144	154 MHz	641	999 LUTs
	Spartan-6	xc6slx4-3tqg144	215 MHz	649	781 LUTs
	Virtex-4	xc4vfx12-12-sf363	218 MHz	641	1005 LUTs
	Virtex-5	xc5vlx20t-2-ff323	221 MHz	638	921 LUTs
	Virtex-6	xc6vcx75t-2-ff484	290 MHz	649	757 LUTs
	Kintex-7	xc7k70t-3-fbg676	355 MHz	649	875 LUTs
	Artix-7	xc7a100t-3-csg324	292 MHz	649	765 LUTs
	Virtex-7	xc7vx330t-3-ffg1157	364 MHz	649	873 LUTs