MDIO Master and Slave Controllers

November 2013 Reference Design RD1194

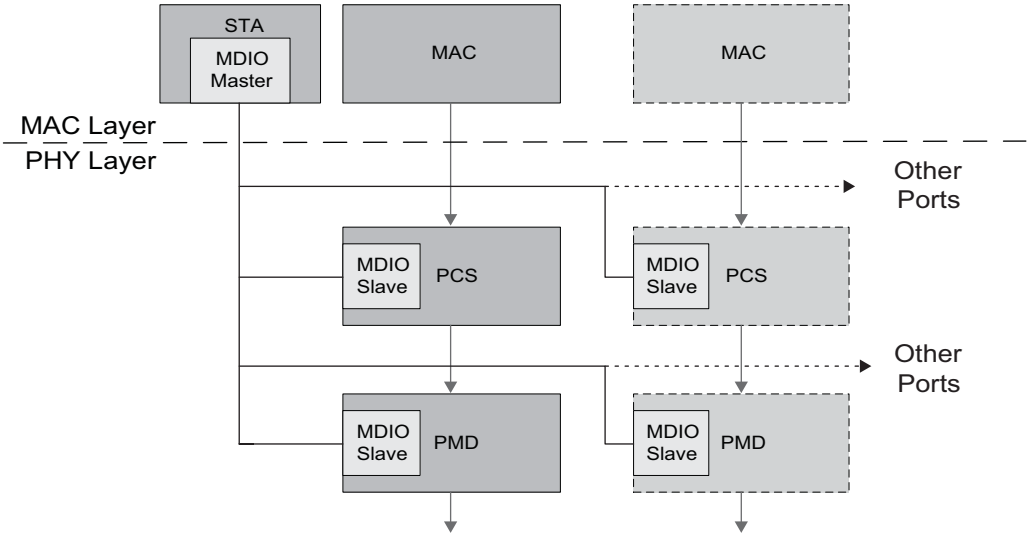
Introduction

Management Data Input/Output Interfaces, or MDIO, are specified in the IEEE 802.3 standard and intended to provide a serial interface to transfer management data between an Ethernet Media Access Controller (MAC) layer and a physical (PHY) layer. The device that controls the MDIO bus is called a Station Management Entity (STA), while the device being managed is called the MDIO Manageable Device (MMD). The STA device is often embedded in the MAC layer and acts as a master for MDIO interface. The MMD is often embedded in the PHY device and is a slave device on MDIO interface. The MDIO interface consists of two pins, a bidirectional MDIO pin and a Management Data Clock (MDC) pin. All data is transferred synchronously to the MDC which is usually provided by the STA or a master controller and sourced to all slave devices. The MDIO is a relatively slow interface running up to 2.5 MHz. However, its ability to access and modify various registers in PHY devices by the master often extends the application beyond the Ethernet system. Figure 1 shows a generic application environment for MDIO interface between MAC layer and PHY layer devices.

IEEE 802.3 Standard specifies two different MDIO interface protocols under Clause 22 and Clause 45. This design provides following MDIO interface compliant cores.

- MDIO master controller which can be used for both clause 22 and clause 45 protocols.
- MDIO slave controller for clause 22 protocol.
- MDIO slave controller for clause 45 protocol.

Figure 1. Generic Application Environment of the MDIO Interface



Features

MDIO Master

- Implements the IEEE 802.3 Standard, Clause 22 and Clause 45 interface
- Simple Wishbone interface for user through register indirect access
- Dynamic selection between clause 22 and clause 45 protocols
- Dynamic selection for Preamble pattern generation in MDIO frames
- Parameterized clock divider for MDC clock

MDIO Slave

- Implements the IEEE 802.3 Standard, Clause 22 and Clause 45 interface
- Simple Wishbone interface for user to implement PHY registers
- PHY address setting through input port for clause 22
- PHY address and Device type settings through input port for clause 45
- Preamble pattern selection through input port for clause 22

Functional Description

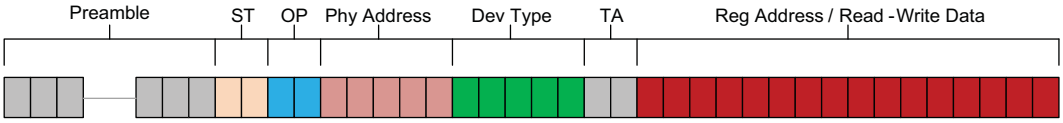
MDIO as originally defined in Clause 22 of IEEE 802.3 specification is able to access up to 32 registers in 32 different PHY devices. The STA initiates all the commands using MDIO frame and provides PHY device address and register address to perform register read or write operation. It also sources clock on pin MDC. MDC is specified to have the frequency up to 2.5Mhz. Figure 2 shows the structure of the MDIO frame used for these operations. The frame format allows only 5bits of PHY address and 5bits of Register address which limits to 32 PHY devices and 32 registers in each device.

Figure 2. Clause 22 Frame Format

Field	Width	Description
Preamble	32 bits	32 bits of “1”s to initialize the transaction.
ST	2 bits	Start of frame (01 for Clause 22)
OP	2 bits	Op code. 01 – Write operation 10 – Read operation
Phy Address	5 bits	PHY (port) address.
Reg Address	5 bits	Register address
TA	2 bits	Turnaround time for Slave to start driving read data if read operation.
Read-Write data	16 bits	Data driven by master in write operation and driven by slave in read operation.

To address the limitations of Clause 22, the Clause 45 was added to 802.3 specification which extends the existing frame format to give provision to access 32 PHY devices, 32 different device types and up to 64K registers in each of these PHY devices. In Clause 45, Opcode is extended to 4 types. Figure 3 shows the Clause 45 structure of the MDIO frame used of these operations.

Figure 3. Clause 45 Frame Format



Field	Width	Description
Preamble	32 bits	32 bits of “1”s to initialize the transaction.
ST	2 bits	Start of frame (00 for Clause 45)
OP	2 bits	Op code.
		00 – Address frame
		01 – Write frame
		10 – Read frame
		11 – Read+ address increment frame
Phy Address	5 bits	PHY (port) address.
Device Type	5 bits	Register address
TA	2 bits	Turnaround time for Slave to start driving read data if read operation.
Reg address/ Read-Write data	16 bits	Data driven by master for “Address”, “Write” frames and driven by slave for “Read”, “Read+address increment” frames.

MDIO Master

MDIO master core is provided as a Verilog source code in “source/mdio_mster.v”. Figure 4 shows a functional block diagram of the MDIO master core. The user interface is provided as a simple Wishbone compliant interface. Table 1 provides the details of these I/Os.

Figure 4. MDIO Master Functional Diagram

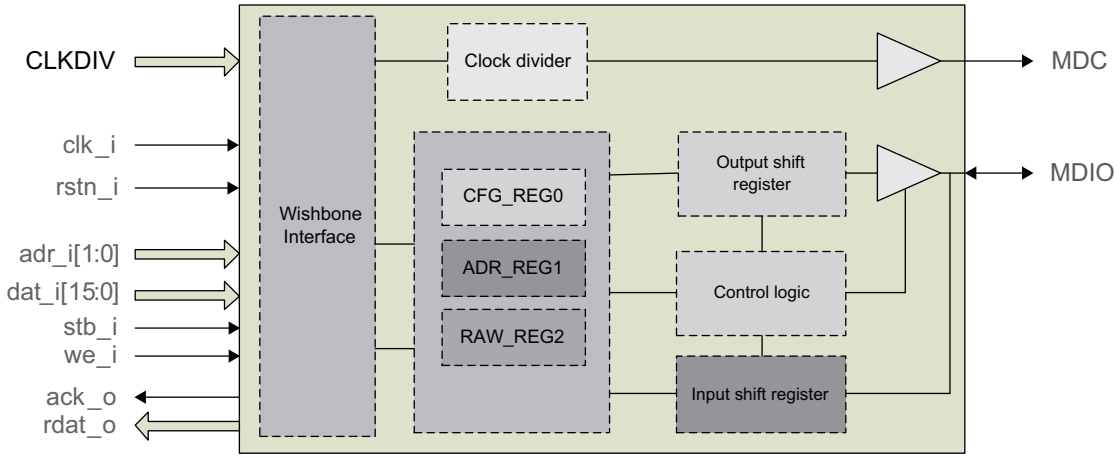


Table 1. MDIO Master Signal Description

Signal / Parameter	Type	Description
CLKDIV	Input	Parameter for MDC clock frequency
Wishbone Interface		
clk_i	Input	Wishbone interface clock
rstn_i	Input	Asynchronous reset
adr_i[1:0]	Input	Address bus to the core
tga_i	Input	Used only for Clause 45. Address tag. 1'b1 along with "adr_i" generates additional address frame on MDIO interface
dat_i[15:0]	Input	Data towards the core
we_i	Input	Write enable input
stb_i	Input	Strobe signal/core select input
rdat_o[15:0]	Output	Data from the core
ack_o	Output	Bus cycle acknowledge
MDIO Interface		
MDC	Output	MDIO interface clock
MDIO	Bidi	Bidirectional MDIO interface data

The master core generates the clock MDC using input clock "clk_i" from the Wishbone interface. The parameter "CLKDIV" can be used to define the frequency relationship between input "clk_i" and output "MDC". The MDC clock frequency is the frequency of "clk_i" divided by the "CLKDIV" parameter value. For an example of 100MHz of "clk_i" clock, CLKDIV parameter setting of "40" gives MDC clock of 2.5MHz. The minimum possible value of on CLKDIV parameter is 4.

The master core implements indirect addressing mechanism to access PHY registers across MDIO interface. It implements three 16bit wide registers which can be read and written from the Wish bone interface.

The field details of these 3 register are shown in Figure 5. The register "CFG_REG0" at address 2'b00 and register "ADR_REG1" at address 2'b01 of the wishbone address space when MDIO master is selected, can be written with different fields which are used for building the MDIO frames. There are no MDIO frames generated when these registers are accessed. The MDIO frame is generated on serial "MDIO" pin only when the register "RAW_REG2" is accessed.

The MDIO master core can be used generate both Clause 22 and Clause 45 frame formats. Setting the register bit CFG_REG[15] to 1'b1 makes the core as Clause 22 master and setting to 1'b0 makes the core as Clause 45 master.

Figure 5. MDIO Master Registers for Wishbone Access

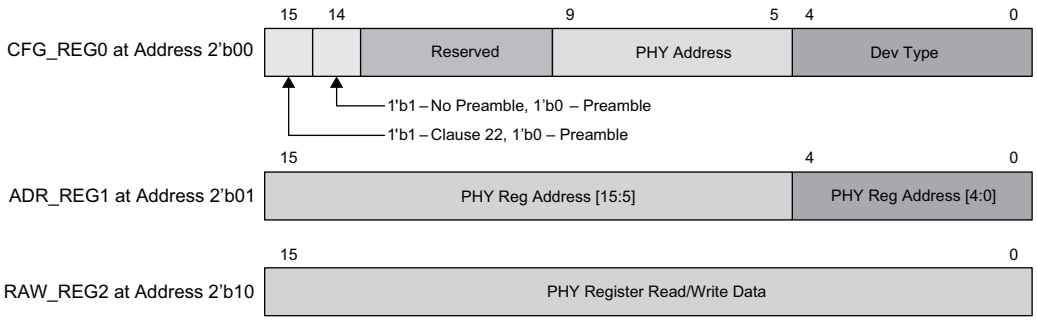


Table 2. MDIO Master Registers for Wishbone Access.

Register Address	Register Fields	Field Name	Default	Description
CFG_REG0				
2'b00	15	cls22	1'b0	Selects between Clause 22 and Clause 45. 1'b0 – Clause 45 1'b1 – Clause 22
	14	no_pre	1'b0	Used only for Clause 22. Selects if preamble is required in generated MDIO frames. Preamble is always generated for Clause 45 frames. 1'b0 – Preamble is required. 1'b1 – Preamble is not required.
	13:10	Reserved	4'b0000	
	9:5	Phy address	5'b00000	5 bit PHY Address field, used for both Clause 22 and 45.
	4:0	Device type	5'b00000	5 bit Device Type field, used for both only Clause 45.
ADR_REG1				
2'b01	15:5	Phy Reg address[15:5]	11'h000	Higher [15:5] bits of PHY register address field, used for both only Clause 45.
	4:0	Phy Reg address[4:0]	5'b00000	Lower [4:0] bits of PHY register address field, used for both Clause 22 and 45.
RAW_REG2				
2'b10	15:0	Phy Reg read write data	-	Read or write data register. An access to this register generates corresponding MDIO frames.

MDIO core is used as Clause 22 master

When there is read access to "RAW_REG2" there will be an MDIO frame sent out on serial line with "op code" set to "10"(read) and all other fields set in accordingly taken from the fields in "CFG_REG0" and "ADR_REG1". At the end of wishbone read cycle the 16 bit data is provided back on wishbone interface which was received from remote MDIO slave through MDIO frame. Similarly, when there is write access to "RAW_REG2" there will be MDIO frame sent out on serial line with "op code" set to "01"(write) and the 16 bit data provided in this wishbone write cycle is used as the data in the generated MDIO frame.

MDIO core is used as Clause 45 master

It also uses the "tga_i" input of the core along with the "adr_i" input from the wishbone interface. Following four types of MDIO frames are generated.

- *read access to "RAW_REG2" with "tga_i" as 1'b0*: MDIO frame with "op code" set to "10"(read+address increment) is generated and at the end of wishbone read cycle the 16 bit data is provided back on wishbone interface which was received from remote MDIO slave through MDIO frame.
- *write access to "RAW_REG2" with "tga_i" as 1'b0*: MDIO frame with "op code" set to "01"(write) is generated and the 16 bit data provided in this wishbone write cycle is used as the data in the generated MDIO frame.
- *read access to "RAW_REG2" with "tga_i" as 1'b1*: There will be 2 MDIO frames generated. 1). MDIO frame with "op code" set to "00"(address) with 16 bit address value from "ADR_REG1" is used. 2). MDIO frame with "op code" set to "11"(read) is generated and at the end of wishbone read cycle the 16 bit data is provided back on wishbone interface which was received from remote MDIO slave through MDIO frame.
- *write access to "RAW_REG2" with "tga_i" as 1'b1*: There will be 2 MDIO frames generated. 1). MDIO frame with "op code" set to "00"(address) with 16 bit address value from "ADR_REG1" is used. 2). MDIO frame with "op code" set to "01"(write) is generated and the 16 bit data provided in this wishbone write cycle is used as the data in the generated MDIO frame.

The following steps are involved in making different types of access from an MDIO master.

Clause 22 Read and Write

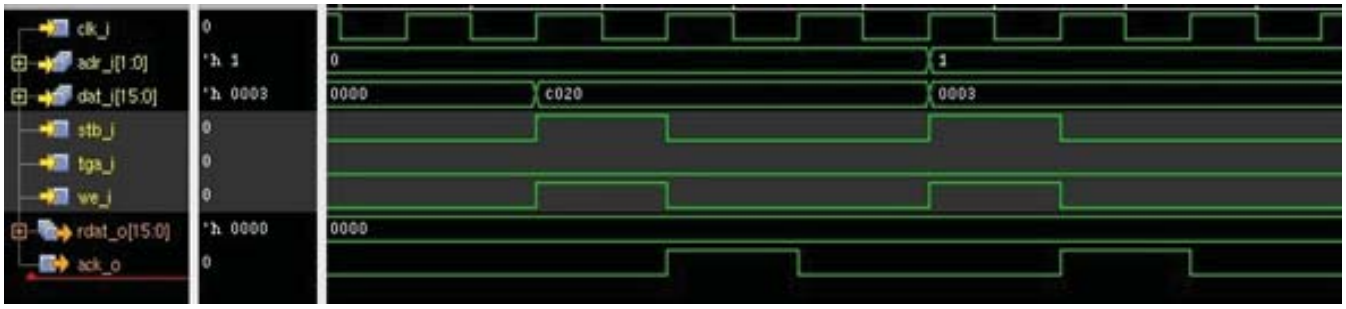
- Write to CFG_REG0, with bit “cls22” set to 1’b1 and required “Phy address” field.
- Write to ADR_REG1[4:0], with required register address.
- Read from RAW_REG2 to read from an MDIO slave PHY register corresponding to address in register ADR_REG1.
- Write to RAW_REG2 to write to an MDIO slave PHY register corresponding to address in register ADR_REG1

Clause 45 Read and Write

- Write to CFG_REG0, with bit “cls22” set to 1’b0 and required “Phy address”, “Device type” fields.
- Write to ADR_REG1[15:0], with required register address.
- Read from RAW_REG2 to read from an MDIO slave PHY register corresponding to address in register ADR_REG1.
- Write to RAW_REG2 to write to an MDIO slave PHY register corresponding to address in register ADR_REG1
- Use “tga_i” input accordingly to generate MDIO address frames depending upon the register address used in the previous command frame like “read+address increment”.

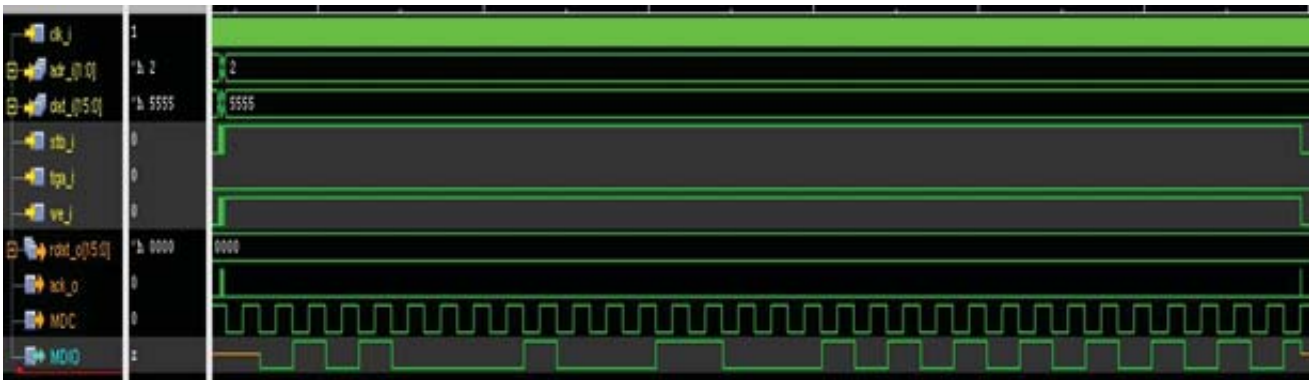
Following Figure 6 shows the timing diagram of the wishbone write cycle access to register CFG_REG0 at address 2’b00 followed by another wishbone write cycle access to register ADR_REG1 at address 2’b01

Figure 6. Wishbone Write Access to Register CFG_REG0 and ADR_REG1



Following Figure 7 shows the timing diagram of the wishbone write cycle access to register RAW_REG2 at address 2’b10 followed which generates an MDIO frame on MDIO/MDC lines.

Figure 7. Wishbone Write Access to Register RAW_REG2 to Generate MDIO Frame



MDIO Slave

MDIO slave core is provided as a Verilog source code in “source/mdio_slave22.v” for Clause 22 and “mdio_slave45.v” for Clause 45. Figure 8 shows a functional block diagram of the MDIO slave 22/45 master core. The user interface is provided as a simple Wishbone interface. Table 3 provides the details of these I/Os.

Figure 8. MDIO Slave Functional Diagram

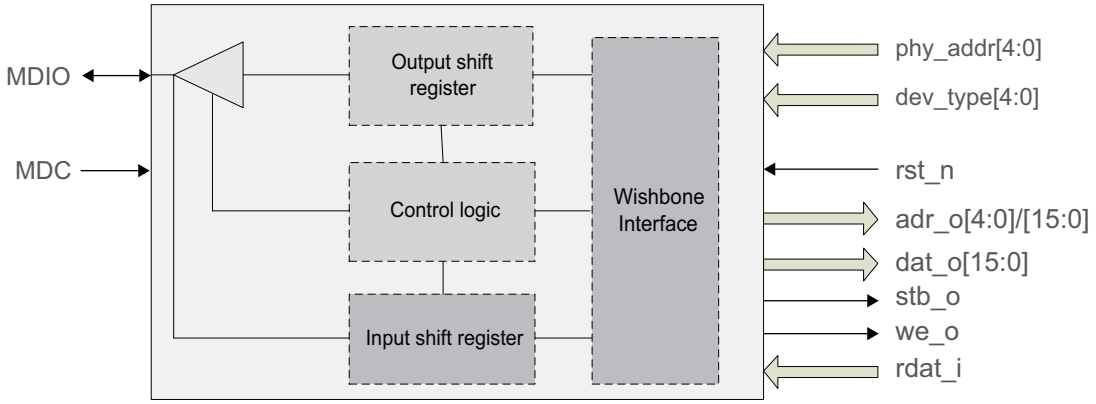


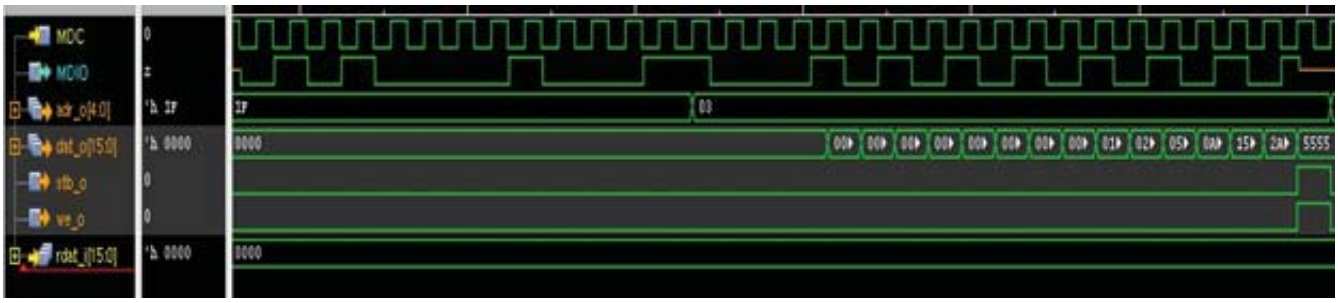
Table 3. MDIO Slave Signal description

Signal	Type	Description
phy_addr[4:0]	input	PHY address to be claimed by this slave. Used for both “mfio_slave22” and “mdio_slave45”.
dev_type[4:0]	input	Device type to be claimed by this slave. Used for only in “mdio_slave45”.
no_pre	input	Only in “mdio_slave22”. Selects if preamble is present in receiving MDIO frames.
		1’b0 – Preamble is required.
		1’b1 – Preamble is not required.
Wishbone Interface		
rst_n	Input	Asynchronous reset
adr_o[4:0]/[15:0]	Output	Address bus to Registers. The bus is [4:0] in “mdio_slave22 and [15:0] in “mdio_slave45.
dat_o[15:0]	Output	Data to write into the registers.
we_o	Output	Write enable output
stb_o	Output	Strobe signal/registers select output
rdat_i[15:0]	Input	Data from the registers, sampled on next positive MDC clock edge after “stb_o” is asserted.
MDIO Interface		
MDC	Input	MDIO interface clock
MDIO	Bidi	Bidirectional MDIO interface data

The slave core is implemented in two separate control logic modules. The module “mdio_slave22” is for Clause 22 and “module_slave45” is for Clause 45 implementations. These modules implement only control state machine and required input and output shift register logic to interface with 2-line MDIO interface and provides a simple Wishbone interface for the register implementation. The slave cores don’t implement actual registers as mentioned in 803.2 Clause 22 and clause 45 specifications. It is required for the user to implement those registers corresponding to the user’s type of the PHY layer application device. Along with mdio_slave22/45” modules “mdio_slave_ref22/45” are provided as example usage of these slave modules along with some read-only, read-write registers.

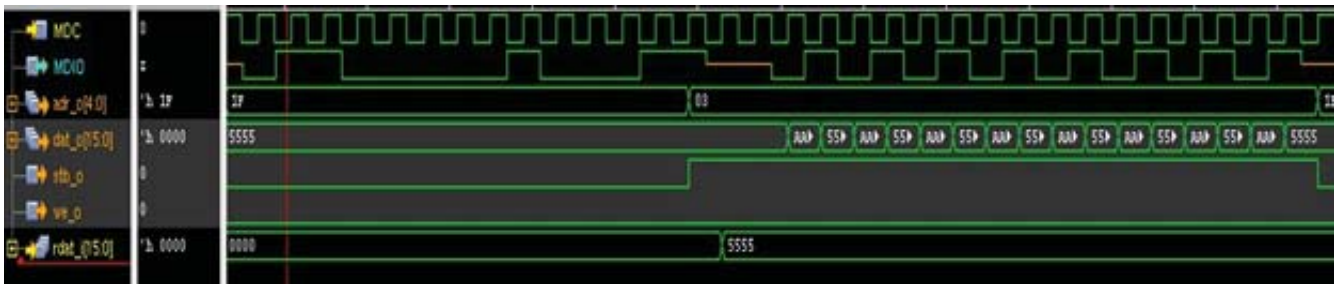
Following Figure 9 shows the timing diagram of the wishbone write cycle access to user register generated after receiving an MDIO write frame from MDIO/MDC lines.

Figure 9. MDIO Slave Write Cycle to Register



Following Figure 10 shows the timing diagram of the wishbone read cycle access to user register generated after receiving an MDIO write frame from MDIO/MDC lines.

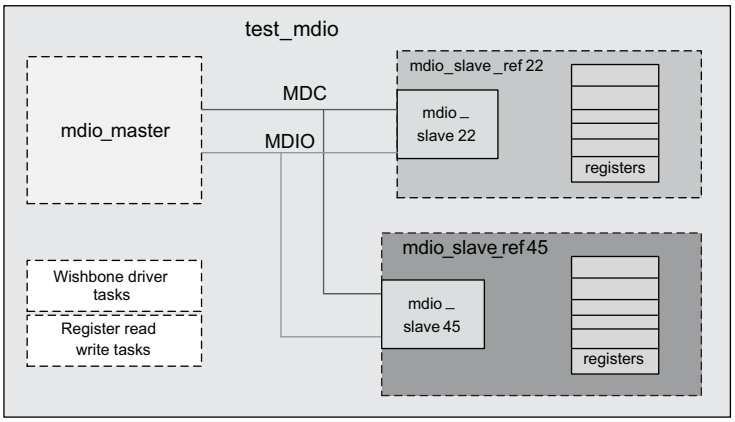
Figure 10. MDIO Slave Read Cycle to Registers



Test Bench and Simulation

Along with mdio_master and mdio_slave cores, an example test bench and required scripts are provided to run evaluation functional simulations. Figure 11 shows a block diagram of this test bench.

Figure 11. Test Bench



The test bench uses the following files:

- **testbench/test_mdio.v** – Top-level test bench file that instantiates a “mdio_master”, “mdio_slave_ref22” and “mdio_slave_ref45”. It also contains the logic implemented for Wishbone driver tasks and few example wishbone access to the mdio master to access remote mdio slave registers.
- **simulation/runsim_mdio_aldec.do** – script file to run this functional simulation using Aldec simulator.
- **simulation/runsim_mdio_modelsim.do** – script file to run this functional simulation using ModelSim simulator.

Implementation

This design is implemented in Verilog. When using this design in a different device, density, speed, or grade, performance and utilization may vary. Default settings are used during in diamond software for the implementation of the design. The “impl” directory provides example .ldf and .lpf files for each of “mdio_master”, “mdio_slave_ref22” and “mdio_slave_ref45” cores which can be used for implementation using Lattice Diamond software.

Table 4. Performance and Resource Utilization for mdio_master

Family	Speed Grade	LUTs	Registers	Frequency (MHz)
ECP3	-6	137	106	>130
XO2	-4	135	103	>100

Table 5. Performance and Resource Utilization for mdio_slave22

Family	Speed grade	LUTs	Registers	Frequency (MHz)
ECP3	-6	96 / 198[1]	76 / 297[1]	>130
XO2	-4	68 /169[1]	67 / 180[1]	>65

1. Uses an implementation of example registers.

Table 6. Performance and Resource Utilization for mdio_slave45

Family	Speed grade	LUTs	Registers	Frequency (MHz)
ECP3	-6	124 / 372[1]	93 / 330[1]	>80
XO2	-4	124 / 299[1]	83 / 307[1]	>50

1. Uses an implementation of example registers.

References

IEEE 802.3 Standard specification.

Technical Support Assistance

e-mail: techsupport@latticesemi.com

Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
November 2013	01.0	Initial release.