ASIC IMPLEMENTATION OF I2C MASTER CONTROLLER
Using Verilog HDL

Table of Contents:

1. Introduction:

The I2C Bus is a two wire serial communication protocol. One is SDA(Serial data) and the other is SCL(Serial Clock). The two wires in the I2C bus carry a 7 bit address, R/W control bit, and a data bit. The data (SDA) wire carries the data, while the clock (SCL) wire synchronizes the sender and receiver during transfer. Each device has its own unique address, and can work either as a transmitter or as a receiver. I2C master is the device that initiates communication and generates a clock for the same. Any device addressed by the master is called a slave. The I2C bus has three modes of operation: standard mode (0 to 100kbps), fast mode (0 to 400kbps), and high-speed mode (0 to 3.4mbps).

2. I2C Protocol Working:

Master will issue an 'Attention' signal to all of the connected devices known as START. Then the Master sends the ADDRESS of the device it wants to communicate with and a Read or a Write bit. All devices compare it with their own address and the one whose address is matched will produce a response signal as an Acknowledgement. If it doesn't match, they simply wait until the bus is released. On receiving acknowledgement, the master can start transmitting or receiving DATA. The transferred data is 8 bits long followed by an acknowledgement bit which is repeated till the whole transmission takes place. When all is done, the Master will issue the STOP condition.

2.1 General Characteristics:

2.1.1 Write Operation: It is done when the R/W bit is 0.

2.1.2 Read Operation: It is done when the R/W bit is 1.

2.1.3 Start and Stop condition: Start bit is indicated by the high-to-low transition of SDA while the SCL keeps high. And the stop bit is indicated by a low-to-high transition on the SDA line keeping SCL high and rest of the transitions of SDA line take place with SCL low.

2.1.4 Data Validity: The data on the SDA line is valid for the high period of clock pulse.

2.1.5 Transmitting a byte to a slave device: After the start condition, a byte transmitted will identify the slave on the bus and will select the mode of operation. If the R/W bit in the address was set to 0, transmission of byte is done.

2.1.6 Receiving a byte from slave device: Once the slave has been addressed and the slave has acknowledged this, a byte can be received from the slave if the R/W bit in the address was set to 1.

2.2 Operation Algorithm:

1. Master generates a START signal and all the slave devices listen to the bus.

2. Master sends a slave device ADDRESS + R/W bit.

3. Slave sends an ACKNOWLEDGE bit to the master if the address is matched.

4. The R/W bit is checked and the direction of data transfer is decided.

5. Slave sends 8-bit DATA to the master when R/W bit is 1 and receives 8-bit DATA from the master when R/W bit is 0.

6. An ACKNOWLEDGE signal is sent after receiving each 8-bit DATA by the receiver unit.

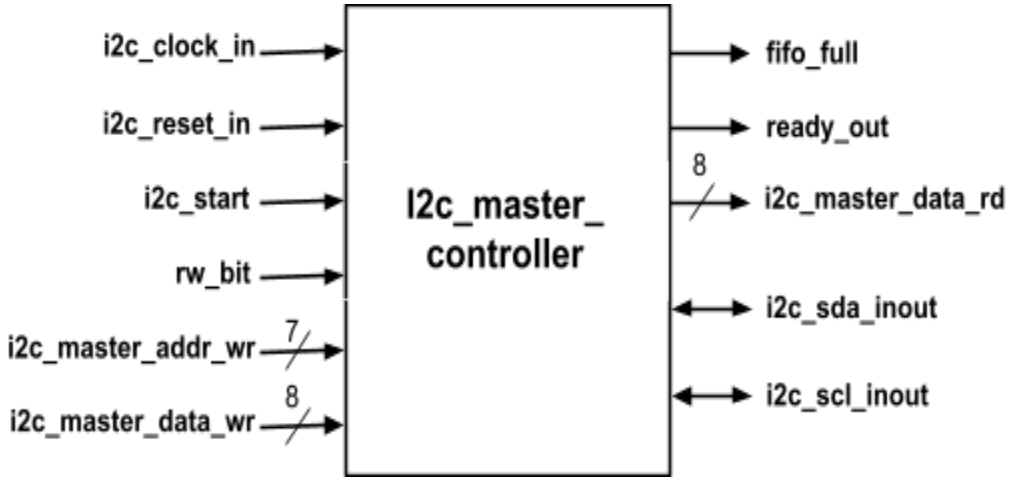
7. The DATA is repeatedly sent or received if there is no STOP signal.

8. The Master sends a STOP signal if data transmission is done.

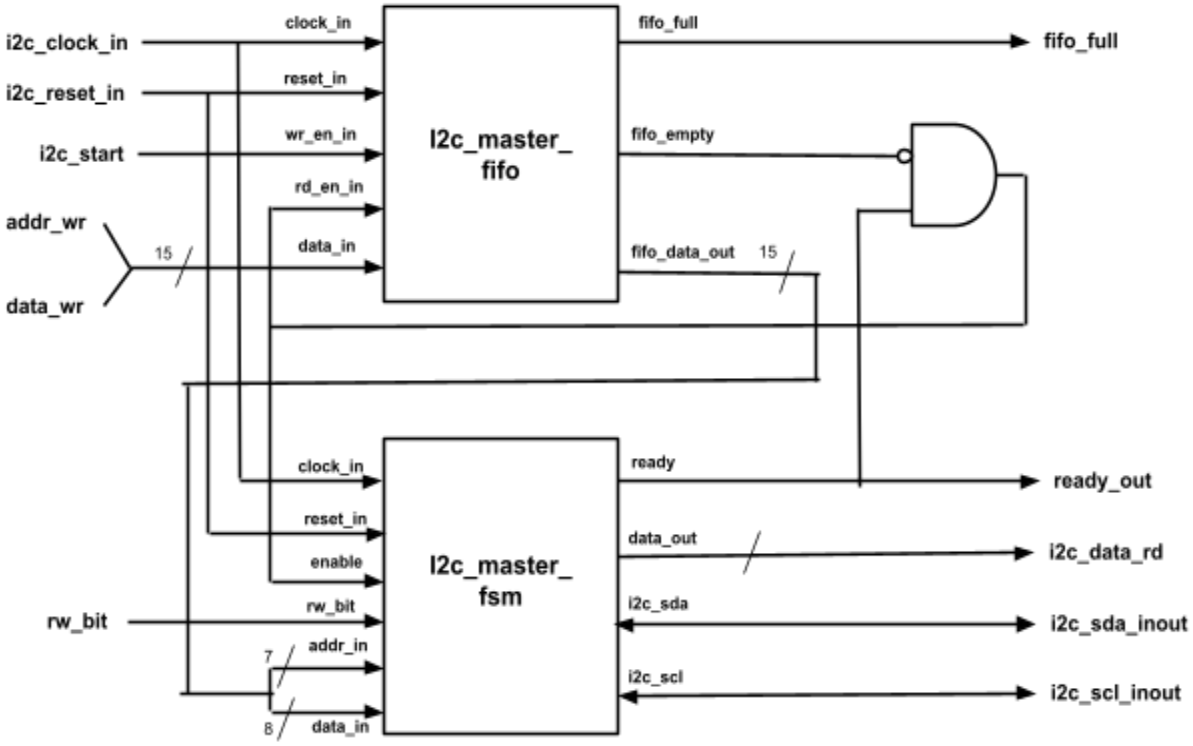
3. I2C Master Controller:

It is the top module which encapsulates a FIFO and a FSM. The FIFO and the FSM are sequential logic and are synchronous in operation with each other, which means both the sub-modules work with the same clock frequency. It allows proper communication between the submodules such as FIFO and FSM.

3.1 Block Diagram:



3.3 Micro Architecture of I2C Master Top:



This top module Micro Architecture is the hierarchical architecture of two lower level modules i2c_master_fifo and i2c_master_fsm, and a logic circuit.

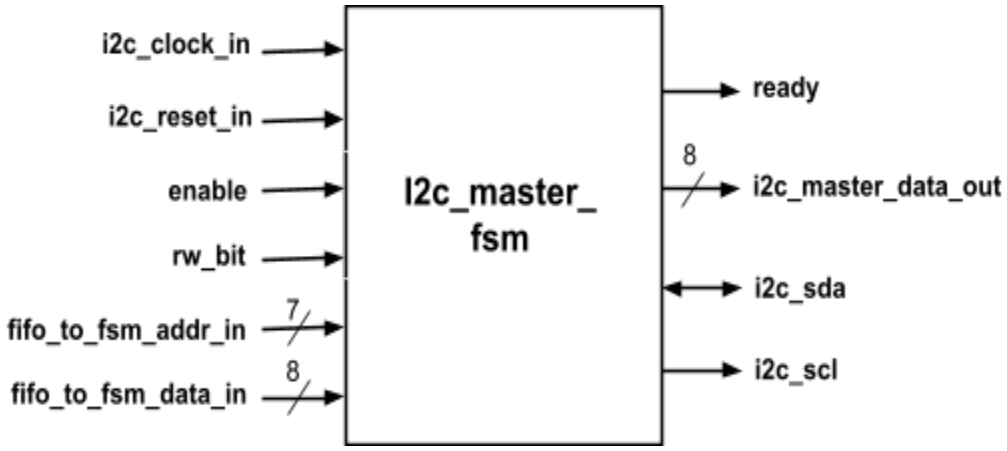
3.2 Port Description:

Signals	Declaration	Description
i2c_clock_in	input	Top module clock signal
i2c_reset_in	input	Active high reset signal
i2c_start	input	If high it enables FIFO write operation
rw_bit	input	Used for read and write operation(1 for read and 0 for write)
i2c_master_addr_wr	input	7bit address in which master read & write data
i2c_master_data_wr	input	1 Byte of data write into slave
i2c_master_data_rd	output	1 Byte of data read from slave
fifo_full	output	Shows high when fifo is full
ready_out	output	Master is ready to transfer addr & data
i2c_sda_inout	inout	Serial data line
i2c_scl_inout	inout	Serial clock line

4. I2C Master FSM:

FSM is the main component of the I2C Master Controller. The FSM is defined as a sequential circuit which uses the finite number of states to track the history of operations. Nine states are present in the FSM to get the actual working of the I2C Master Controller.

4.1 Block Diagram:



4.2 Port Declaration:

Signals	**Declaration**	**Description**
i2c_clock_in	input	Top module clock signal
i2c_reset_in	input	Active high reset
enable	input	0: no transaction is initiated. 1: latches addr, rw, & data to initiate a transaction.(latches addr, rw, & data in next start state)
rw_bit	input	Used for read and write 0: write operation 1: read operation

fifo_to_fsm_addr_in	input	Upper 7 bits of fifo_data_out[14:8] are the addresses connected to the i2c_master_fsm address port. Through this address+rw_bit master is read and write data.
fifo_to_fsm_data_in	input	Lower 8 bits of fifo_data_out[7:0] are connected to the i2c_master_fsm data port. This 8 bit data is written into slave when rw_bit is 0.
i2c_master_data_out	output	Connected to i2c_master_controller data_rd output port. When rw_bit is high it read 8 bit data from slave
ready	output	Connected to the ready_out port of i2c_master_controller. It ensures the master is ready to transfer addr and data.
i2c_sda	inout	Serial data line
i2c_scl	output	Serial clock line

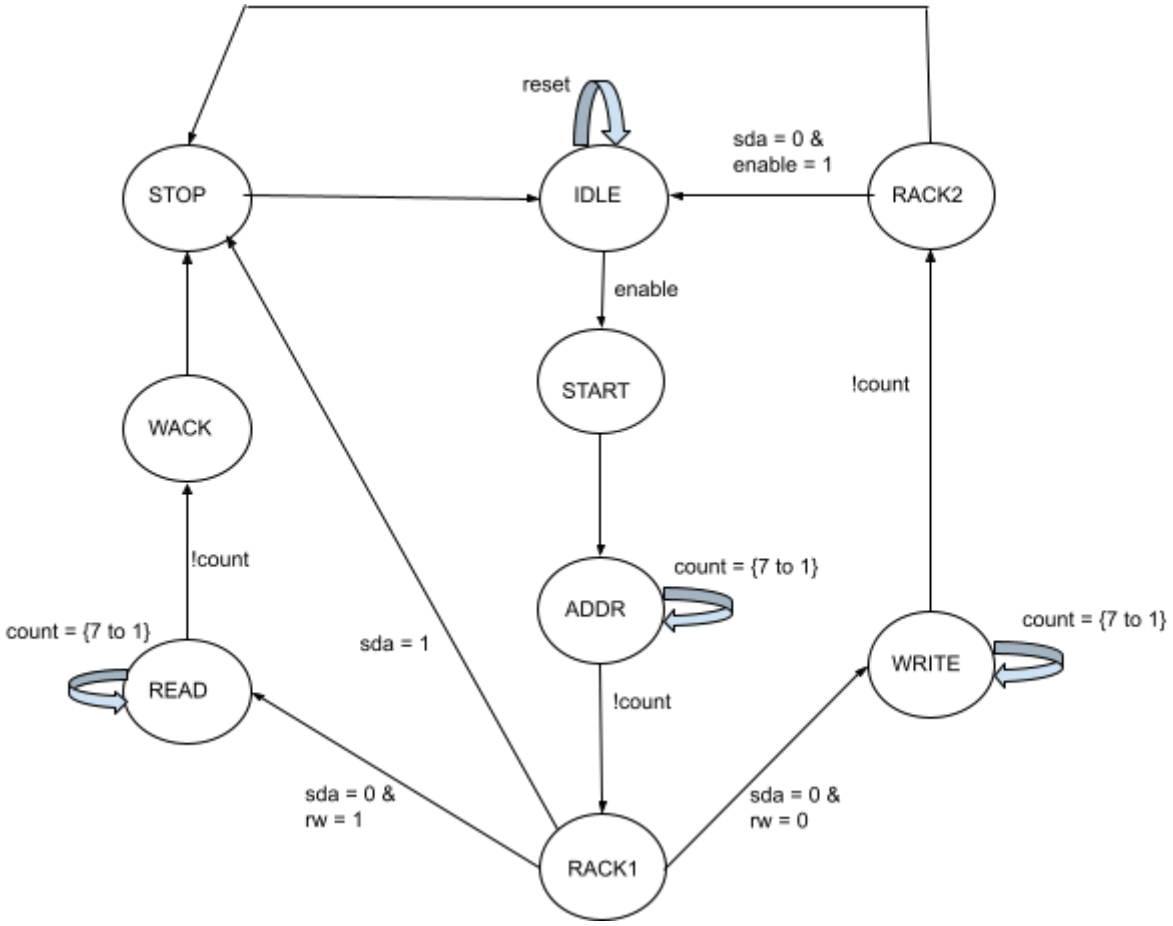
4.3 Functional Description:

Algorithm:

State1: *STATE_IDLE* : I2C Bus does not perform any operation (i.e SCL and SDA remains high) and enable is low. If enable is high it enters into the START state.
State2: *STATE_START* : Master initiates data transmission by entering into the next state STATE_ADDR and by providing START (SCL is high and SDA goes high to low).
State3: *STATE_ADDR* : In this ADDR state, Master sends the slave address serially to the slave. Counter is used to count no of bits transferred and as it becomes 0, it enters into next state STATE_RACK1.
State4: *STATE_RACK1* : In this state Slave sends an acknowledgement bit to Master by pulling SDA line to low(here we have used Slave RAM so no need to check Slave_address).
If ack_bit is received then it checks with rw_bit, if rw_bit is high it enters into the READ state, else enters into the WRIGHT state.
If ack is not received it enters into IDLE state.

State5: *STATE_WRITE* : In this state, the 8 bit data to be transmitted is sent to the slave by the master. After receiving data Slave acknowledges the Master. In this also counter is used, when count reaches to 0 it enters into STATE_RACK2 state.
State6: *STATE_RACK2* : In this state, If ack is received and also enable is high then i enters into IDLE state else enters into STOP state.
State7: *STATE_READ* : In this state, the 8 bit data is read from the Slave by the Master. After reading the data it enters into WRITE_ACK state.
Stage8: *STATE_WACK* : In this state, ack is sent from master to slave after reading 8 bits of data and it enters into STOP state.
Stage9: *STATE_STOP* : After the transmission of data or receiving the data a stop bit is sent (SCL is high and SDA goes low to high) and it enters into the IDLE state.

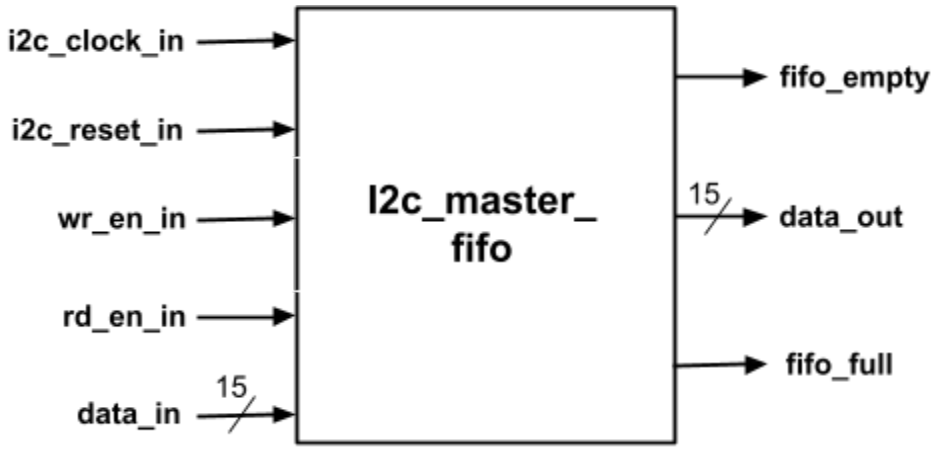
State Machine:



5. I2C Master FIFO:

The width of the FIFO is 15 bits for this proposed design. Whenever the FIFO is full, its filled status is indicated by the status output signal “full” and whenever there are no values stored in the FIFO, its empty status is indicated by the status signal “empty”.

5.1 Block Diagram:



5.2 Port Declaration:

Signals	Declaration	Description
i2c_clock_in	input	Top module clock signal.
i2c_reset_in	input	Active high reset signal.
wr_en_in	input	Enable signal for fifo write operation
rd_en_in	input	Enable signal for fifo read operation
data_in	input	Input data is written into fifo when wr_en is high and fifo is not full.
fifo_full	output	When the fifo is full this signal goes high
fifo_empty	output	When the fifo is empty this signal goes high

data_out	output	Output data read from fifo when rd_en is high and fifo is not empty.

5.3 Functional Description:

Internal counters, rd_ptr, wr_ptr, and fifo_mem are used to design the FIFO. Counter is used to keep the track of fifo_full and fifo_empty flag signals, rd_ptr is used to read the data from fifo_mem to which it points, and wr_ptr is used to write the data into fifo_mem to which it points.

Case1: Write data into fifo

If wr_en_in is high and fifo is not full then fifo_mem[wr_ptr] = data_in.

Case2: Read data from fifo

If rd_en_in is high and fifo is not empty then data_out = fifo_mem[rd_ptr].