FIFO, First-In First-Out Memory

Jamil Khatib

March 28, 1999

1   Top level design:

The First-In First-Out Memory stores the data in queue order so the first
input element goes out the first.

1.1   Design Approach:

My design depends on the use of dual port memory and generation of both
read and write pointers as the addresses for each port.

Depending on this approach my system is divided into four functional
blocks as shown in figure 1. This division can be used by almost all other
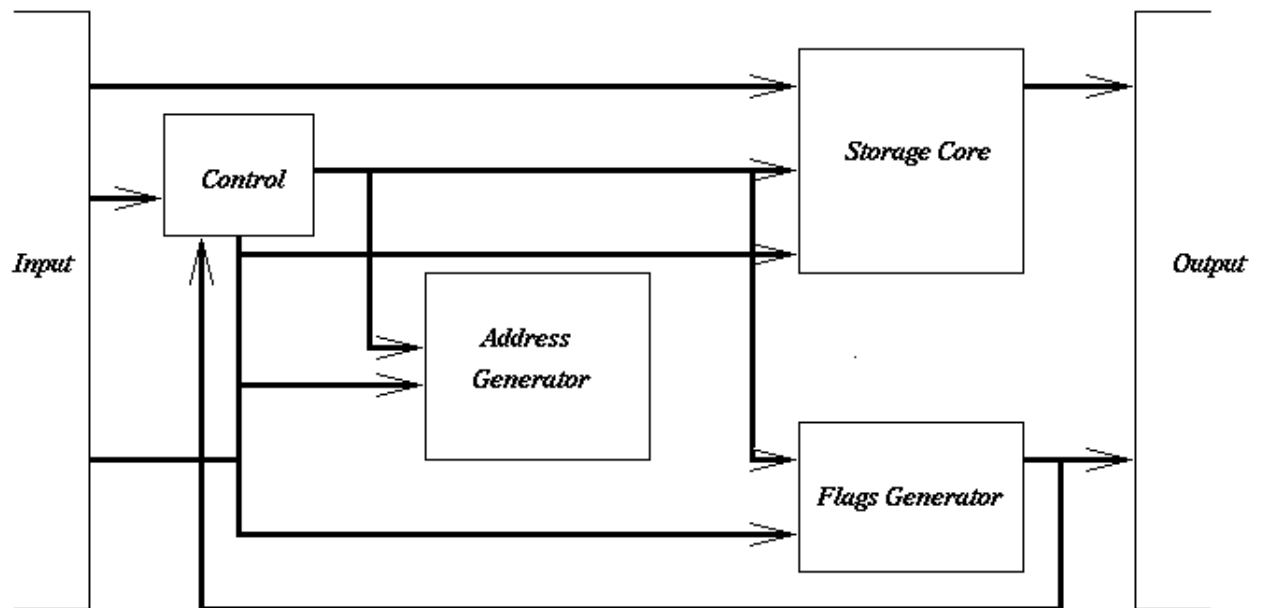approaches that I am going to discuss two of them later.

Figure 1: System Block Diagram

1.2   Interface:

Signal Name	Signal Type	Description	Notes
DATA_IN[7:0]	input bus	input data to be stored	
DATA[7:0]	output bus	output data	
RE	input	read enable signal	Active High
WE	input	write enable signal	Active High
SYSCLK	input	system global clock	
RESET	input	system global reset	Active Low
HALF_FULL	output	half full signal	Active high
FULL	output	full signal no further write attempts are enabled	Active High
EMPTY	output	Empty signal no further read attempts are enabled	Active High

2   Controller Block:

The controller part determines and generates the valid read and write signals depending on both the flags and the system read and write requests.

RE	**EMPTY**	**RE_MEM**
0	0	0
0	1	0
1	0	1
1	1	0

Table 1: Read Operation truth table

WE	**FULL**	**WE_MEM**
0	0	0
0	1	0
1	0	1
1	1	0

Table 2: Write Operation truth table

3   Address generation Block:

This block generates both pointers to the next address to be read and to be written.

3.1 Interface:

Signal Name	Signal Type	Description	Notes
CLK	input	system global clock	
RESET	input	system global reset	Active Low
RE	input	read enable signal	Active High
WE	input	write enable signal	Active High
W_ADDR[7:0]	output bus	The Write pointer	
R_ADDR[7:0]	output bus	The Read pointer	

3.2 Implementation:

- The block is simply implemented of two 8-bit synchronous counters.

- The outputs of these two counters represent the two pointers.

- The **CE** chip enable pins are connected to both the write and read enable that are synchronized with the system read and write requests. These lines should be enabled only on valid read or valid write actions. So external logic should control these lines.

- result the clock goes infnitly to the counters but the count up is performed only upon read and write requests.

- clear pins are connected to the global Reset signal.

4 Dual port memory block:

This block is the storage core of the FIFO.

4.1 Interface:

Signal Name	Signal Type	Description	Notes
CLK	input	system global clock	
RESET	input	system global reset	Active Low
RE_MEM	input	read enable signal	Active High
WE_MEM	input	write enable signal	Active High
WA[7:0]	input bus	The Write address	
RA[7:0]	input bus	The Read address	
DIN[7:0]	input bus	The input data	
DOUT[7:0]	output bus	The output data.	

4.2 Implementation:

- The memory block is generated by the LogiBlox tool. This block is a single and dual port memory with Write enable synchronized to the block clock.

3

- The write operation to the first port is done for each rising edge of the write clock at the address specified for **WA[7:0]** when The **WE_MEM** is active.

- The read operation is not synchronized "Asynchronous read" with the clock and it produces the output data for each input read address **RA[7:0]**

- A 3-state buffer is located at the output of the Dual port output in order to enable and disable the output of the FIFO.

- A Flip-Flop with its input connected to the inverted **RE_MEM** input is to synchronize the Read operation with the block clock and to make it active high signal the same as the write operation since the control of the tri-state buffer is active low.

- A synchronous register is added at the input to synchronize the asynchronous input data to be written.

- A delay Flip-Flop is added at the between the external **WE_MEM** input and the **WR_EN** pin is to synchronize the write command with the input data from the register.

5 The Flags Block:

This block is one of the most problematic blocks. This block generates the required flags that control and validate the read and write signals. It generates three signals, the **FULL** signal, which indicates that the FIFO is Full and so no further write operation should be attempted. The **HALF_FULL** signal which is just an indication that Half the FIFO is full or empty. Finally the **EMPTY** signal that indicates that the FIFO is empty and no further read operation should be attempted unless a single byte is written to the FIFO.

5.1 Interface:

Signal Name	Signal Type	Description	Notes
CLK	input	system global clock	
RESET	input	system global reset	Active Low
RE	input	read enable signal	Active High
WE	input	write enable signal	Active High
FULL	output	Full flag	active high
HALF_FULL	output	half full flag	active high
EMPTY	output	empty flag	active high

4

5.2   Implementation:

- The main goal of this block is to check both the read and write pointers if they are close together and by how much. If the difference between them is zero, then the FIFO is either Full or Empty. To determine if its full or empty the previous state of the pointers must be checked.

- Instead of implementing this idea as is, I decided to use another pointer or indicator that is incremented for each write operation and decremented for each read operation. When this pointer is zero, the FIFO is empty. When the pointer points to the highest address in the memory it means that the FIFO is full.

- Regarding this approach, I used an 8-bit count up/down counter to represent this pointer.

- This counter counts up for valid Write operations

- It counts down for valid read operations

- When there are valid read and write signals at the same time, neither count up nor count down should be done. The same as if there are no valid signals at the input.

- This simple arbitration is controlled using the **CE** "Chip Enable" and **UP** "count UP" pins. The truth table of these functions are described in tables 3 and 4.

RE	WE	CE
0	0	0
0	1	1
1	0	1
1	1	0

Table 3: Arbiter truth table

RE	WE	UP
0	0	X
0	1	1
1	0	0
1	1	X

Table 4: Count direction truth table

So **UP=WE**

- **Signal generation:**
  FULL signal: is generated when all lines of the counter are '1's.
  HALF_FULL signal: is generated when the MSB bit of the counter is '1'.
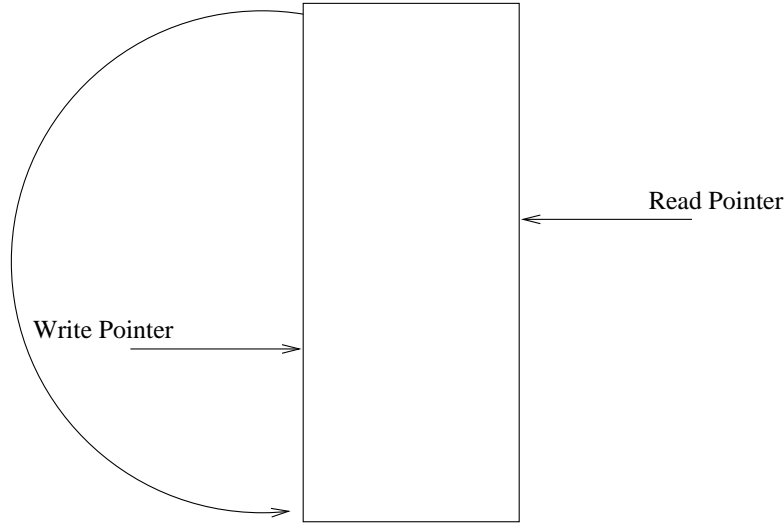 ***EMPTY*** signal: is generated when all lines of the counter are '0's.

Figure 2: Read and write Pinters Block Diagram

6 Improvements and other approaches:

The Flag generation block can be implemented differently by comparing both the Read and Write addresses without adding the Extra counter. In this way the size of the system will decrease.

Another way of implementing the memory core is by using a single port memory and make an arbitration between the read and write operation. This approach will decrease the system size specially for FPGAs and it also decreases the speed of the system.

Array of shift registers can be used to implement the memory core block while a controller is used to control the shift operation. So the input data goes to the first register while the output data goes from the last register. The input data should be shifted down to the last empty register.

The controller can be implemented with self-timed design technique which make the system much faster.