XAPP258 (v1.3) January 17, 2003

FIFOs Using Virtex-II Block RAM

Summary

The Virtex™-II FPGA series provides dedicated on-chip blocks of 18 Kbit True Dual-Port™ synchronous RAM for use in FIFO applications. This application note describes a way to create a common-clock (synchronous) version and an independent-clock (asynchronous) version of a 511 × 36 FIFO, with the depth and width being adjustable within the Verilog or VHDL code.

Introduction

The Virtex-II devices includes fully synchronous dual-ported RAMs with 18K of memory cells. These blocks are ideal for FIFO applications, and each port can be configured independently as 16K × 1, 8K × 2, 4K × 4, 2K × 9, 1K × 18, or 512 × 36.

This application note describes a 511 × 36 FIFO, but each port structure can be changed if the control logic is changed accordingly. The size of the FIFO is 511 × 36 instead of 512 × 36 since one address is dropped out of the FIFO in order to provide distinct Empty/Full conditions. First the design for a 511 × 36 FIFO with common Read and Write clocks is described, and then the design changes required for the more difficult case of independent Read and Write clocks are presented. Signal names in parenthesis are a reference to the name in the Verilog or VHDL code.

Synchronous FIFO Using Common Clocks

Figure 1 is a block diagram of a synchronous FIFO. When both the Read and Write clocks originate from the same source, it simplifies the operation and arbitration of the FIFO, and the Empty and Full flags can be generated more easily. Binary counters are used for both the read (read_addr) and write (write_addr) address counters. Figure 2 shows the timing diagram of a 511 × 36 synchronous FIFO. Table 1 lists the Port Definitions for a synchronous FIFO design.

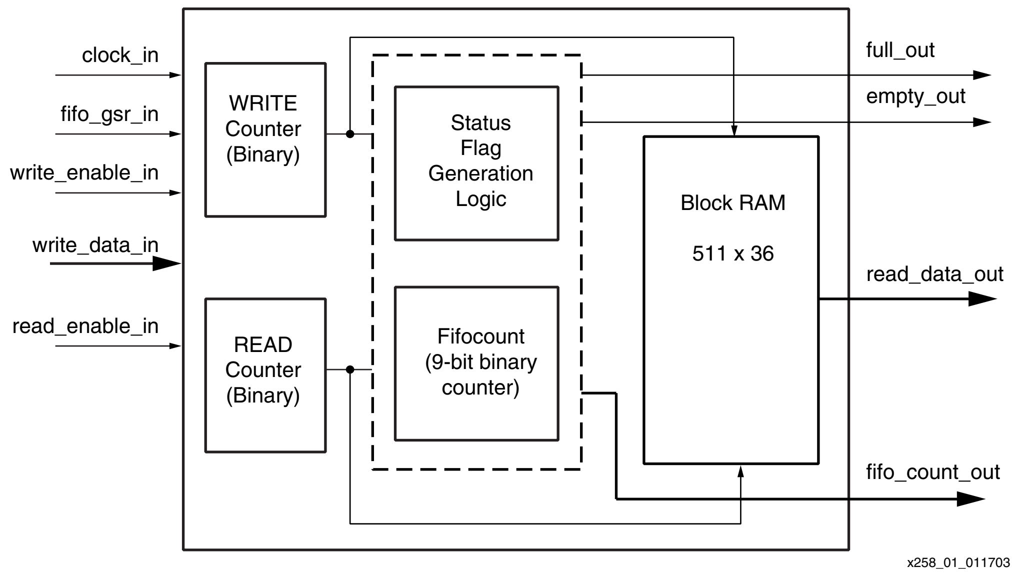


Figure 1: **511 × 36 Synchronous FIFO**

© 2003 Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at http://www.xilinx.com/legal.htm. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

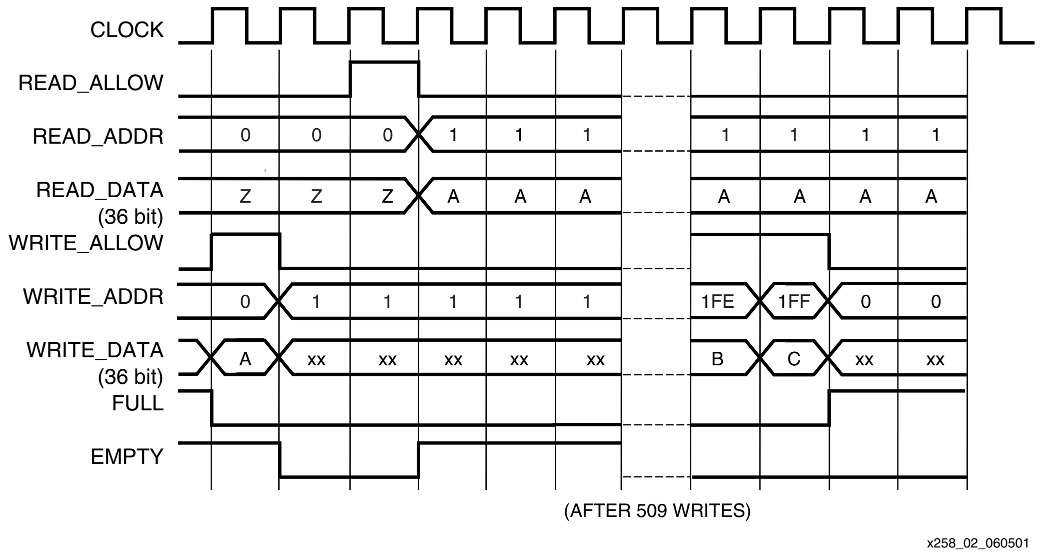


Figure 2: **511 × 36 Synchronous FIFO**

Table 1: **Port Definitions**

Signal Name	Port Direction	Port Width
clock_in	input	1
fifo_gsr_in	input	1
write_enable_in	input	1
write_data_in	input	36
read_enable_in	input	1
read_data_out	output	36
full_out	output	1
empty_out	output	1
fifocount_out	output	4

Synchronous FIFO Operation

To perform a read, Read Enable (read_enable) is driven High prior to a rising clock edge, and the Read Data (read_data) will be presented on the outputs during the next clock cycle. To do a Burst Read, simply leave Read Enable High for as many clock cycles as desired, but if Empty goes active after reading, then the last word has been read, and the next Read Data would be invalid.

To perform a write, the Write Data (write_data) must be present on the inputs, and Write Enable (write_enable) is driven High prior to a rising clock edge. As long as the Full flag is not set, the Write will be executed. To do a Burst Write, the Write Enable is left High, and new Write Data must be available every cycle.

A FIFO count (fifocount) is added for convenience, to determine when the FIFO is 1/2 full, 3/4 full, etc., as shown in Table 3. It is a binary count of the number of words currently stored in the FIFO. It is incremented on Writes, decremented on Reads, and left alone if both operations are performed within the same clock cycle. In this application, only the upper four bits are sent to I/O, but that can easily be modified.

The Empty flag is set when either the fifocount is zero, or when the fifocount is one and only a Read is being performed. This early decoding allows Empty to be set immediately after the last Read. It is cleared after a Write operation (with no simultaneous Read). Similarly, the Full flag is set when the fifocount is 511, or when the fifocount is 510 and only a write is being performed. It is cleared after a Read operation (with no simultaneous Write). If both a Read and Write are done in the same clock cycle, there is no change to the status flags. During Global Reset (fifo_gsr), both these signals are driven High, to prevent any external logic from interfacing with the FIFO during this time.

Asynchronous FIFO Using Independent Clocks

Figure 3 is the block diagram for a 511 × 36 asynchronous FIFO. The asynchronous FIFO Read and Write port signals are clocked by independent Read and Write clocks. Figure 4 shows the timing diagram of a 511 × 36 asynchronous FIFO. Table 2 shows the port definitions for an asynchronous FIFO.

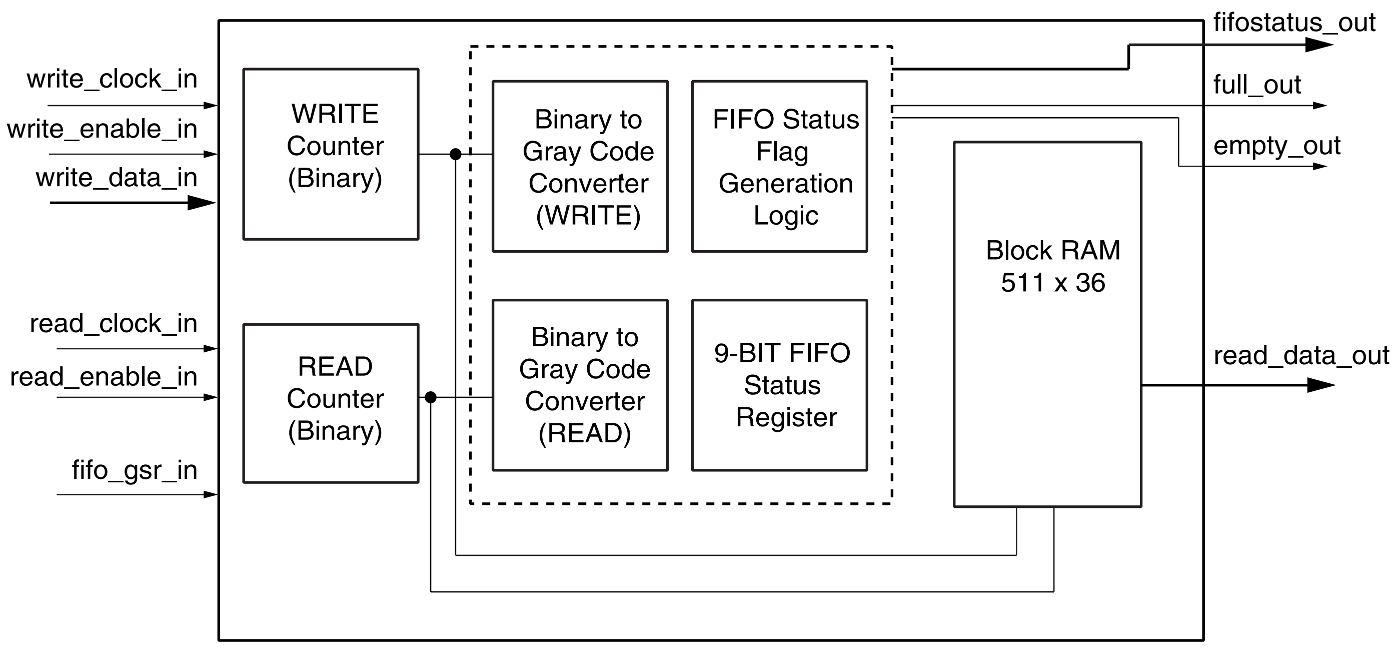
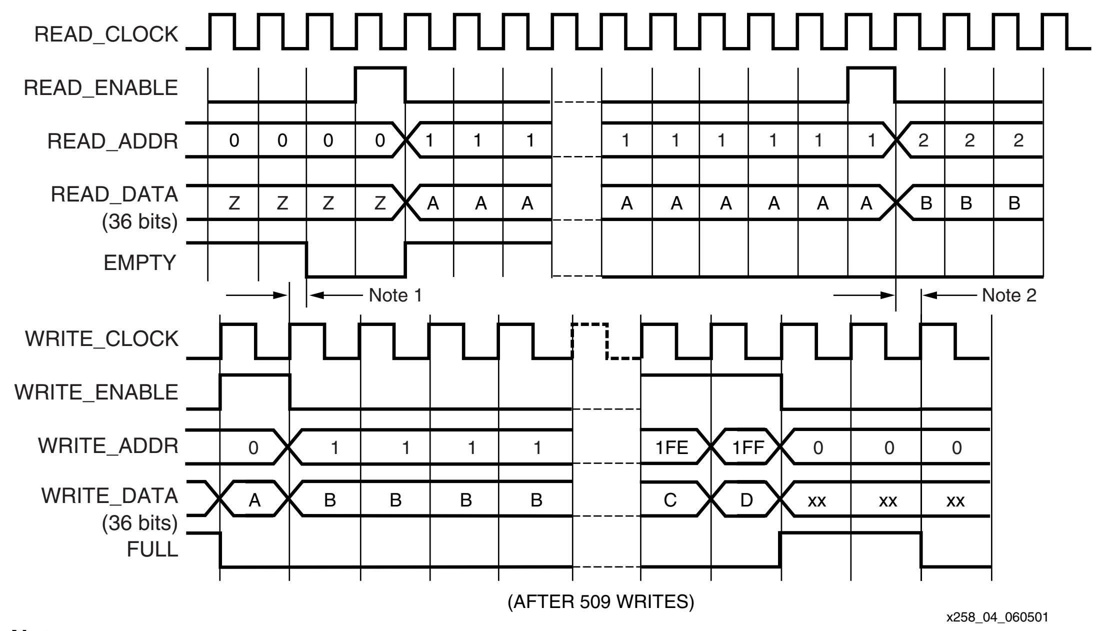


Figure 3: **511 × 36 Asynchronous FIFO**



Notes:

1. Empty will go Low if the write addresses meet the setup time before the rising edge of READ_CLOCK. If not, the Empty will go Low one clock cycle later.
2. Full will go Low if the read addresses meet the setup time before the rising edge of the WRITE_CLOCK. If not, Full will go Low one clock cycle later.

Figure 4: **511 × 36 Asynchronous FIFO**

Table 2: **Port Definitions**

Signal Name	Port Direction	Port Width
write_clock_in	input	1
read_clock_in	input	1
fifo_gsr_in	input	1
write_enable_in	input	1
write_data_in	input	36
read_enable_in	input	1
read_data_out	output	36
full_out	output	1
empty_out	output	1
fifostatus_out	output	4

Asynchronous FIFO Operation

In order to operate a FIFO with independent Read and Write clocks, some asynchronous arbitration logic is needed to determine the status flags. The previous Empty/Full generation logic and associated flip-flops are no longer reliable, because they are now asynchronous with respect to one another, since Empty is clocked by the Read Clock, and Full is clocked by the Write Clock.

To solve this problem, and to maximize the speed of the control logic, additional logic complexity is accepted for increased performance. There are primary 9-bit Read and Write binary address counters, which drive the address inputs to the block RAM. The binary addresses are converted to Gray-code, and pipelined for a few stages to create several address pointers (read_addrgray, read_nextgray, read_lastgray, write_addrgray, write_nextgray) which are used to generate the Full and Empty flags as quickly as possible. Gray-code addresses are used so that the registered Full and Empty flags are always clean, and never in an unknown state due to the asynchronous relationship of the Read and Write clocks. In the worst case scenario, Full and Empty would simply stay active one cycle longer, but this would not generate an error.

When the Read and Write Gray-code pointers are equal, the FIFO is empty. When the Write Gray-code pointer is equal to the next Read Gray-code pointer, the FIFO is full, having 511 (36-bit) words stored. Additional comparisons are done within the same carry chain to determine when the FIFO is Almost Empty and Almost Full, so that Empty and Full can be generated on the same clock edge as the last operation. (Traditional control logic uses an asynchronous signal to set the flags, but this is much slower and limits the overall performance.)

The fifostatus signal indicates 1/2 full, 1/4 full, etc., as shown in Table 3. The task of generating fifostatus in the asynchronous version is more complex, and therefore requires more logic. The overall performance can be improved if this signal is trimmed. The fifostatus outputs have a one-cycle latency for write operations, and a two-cycle latency for reads.

Table 3: **FIFO Count and FIFO Status Signal Descriptions**

Bit 3	Bit 2	Bit 1	Bit 0	FIFOStatus/FIFOCount
1	1	1	1	15/16 full
1	1	1	0	7/8 full
1	1	0	1	13/16 full
1	1	0	0	3/4 full
1	0	1	1	11/16 full
1	0	1	0	5/8 full
1	0	0	1	9/16 full
1	0	0	0	1/2 full
0	1	1	1	7/16 full
0	1	1	0	3/8 full
0	1	0	1	5/16 full
0	1	0	0	1/4 full
0	0	1	1	3/16 full
0	0	1	0	1/8 full
0	0	0	1	1/16 full
0	0	0	0	< 1/16 full

Reference Design

The independent clock design has been bench-tested, and simulation test benches for each of the FIFO designs are provided, and have been simulated using the Model Tech Simulator. The reference design is available on the Xilinx ftp site in both VHDL and Verilog files. Table 4 lists the file names and descriptions for the reference design. (File: xapp258.zip)

Table 4: **Reference Design File Names and Descriptions**

File Name	Description
fifoctlr_cc_v2.v, vhd	common clocks, 511 × 36 FIFO
fifoctlr_ic_v2.v, vhd	independent clocks, 511 × 36 FIFO
tb_x258v_cc.v	testbench for fifoctlr_cc_v2.v
tb_x258v_ic1.v	testbench for fifoctlr_ic_v2.v, Read faster than Write
tb_x258v_ic2.v	testbench for fifoctlr_ic_v2.v, Write faster than Read

Conclusion

The Virtex-II block RAM can be used to generated both synchronous and asynchronous FIFOs. Asynchronous FIFOs are possible due to the true dual-port nature of the block RAM feature. These FIFOs can operate at speeds around 200 MHz.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/09/01	1.0	Initial Xilinx release for Virtex-II series, previous design in XAPP131
02/13/01	1.1	Updated Figure 2.
06/05/01	1.2	Updated Figure 2 and Figure 4.
01/17/03	1.3	Updated template and fixed a typo in Table 3.