XAPP131 (v1.3) February 2, 2000

170 MHz FIFOs Using the Virtex Block SelectRAM+ Feature

Summary

The Virtex™ FPGA Series provides dedicated on-chip blocks of 4096 bit dual-port synchronous RAM, which are ideal 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 x 8 FIFO, with the depth and width being adjustable within the Verilog code. A hand-placed version of the design runs at 170MHz in the -6 speed grade.

Introduction

The Virtex Series of devices includes Block SelectRAM+™, which are fully synchronous dual-ported RAMs with 4096 memory cells. These blocks are ideal for FIFO applications, and each port can be configured independently as 4K x 1, 2K x 2, 1K x 4, 512 x 8 or 256 x 16.

This application note describes a 511 x 8 FIFO, but each port structure can be changed if the control logic is changed accordingly. The size of the FIFO is 511 x 8 instead of 512 x 8 since one address is dropped out of the FIFO in order to provide distinct EMPTY/FULL conditions. First the design for a 511 x 8 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 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 x 8 synchronous FIFO. Table 1 is the Port Definitions for a synchronous FIFO design.

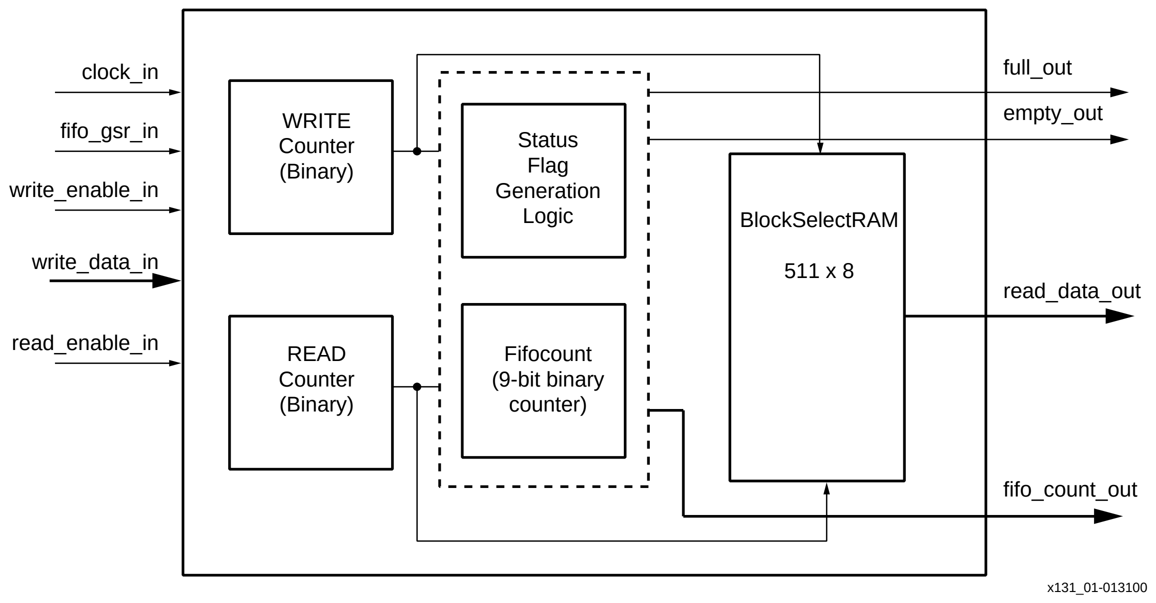


Figure 1: **511 x 8 Synchronous FIFO**

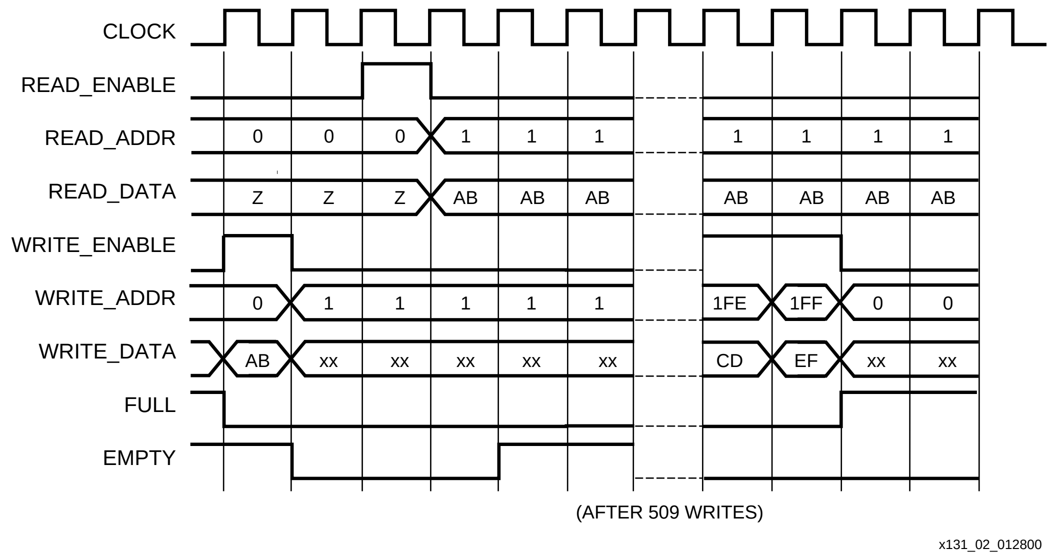


Figure 2: **511 x 8 Synchronous FIFO**

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	8
read_enable_in	input	1
read_data_out	output	8
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.

The Empty flag is set when the Next Read Address (next_read_addr) is equal to the current Write Address, 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 Next Write Address (next_write_addr) is equal to the current Read Address, 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.

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

Asynchronous FIFO Using Independent Clocks

Figure 3 is the block diagram for a 511 x 8 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 x 8 asynchronous FIFO. Table 2 shows the port definitions for an asynchronous FIFO.

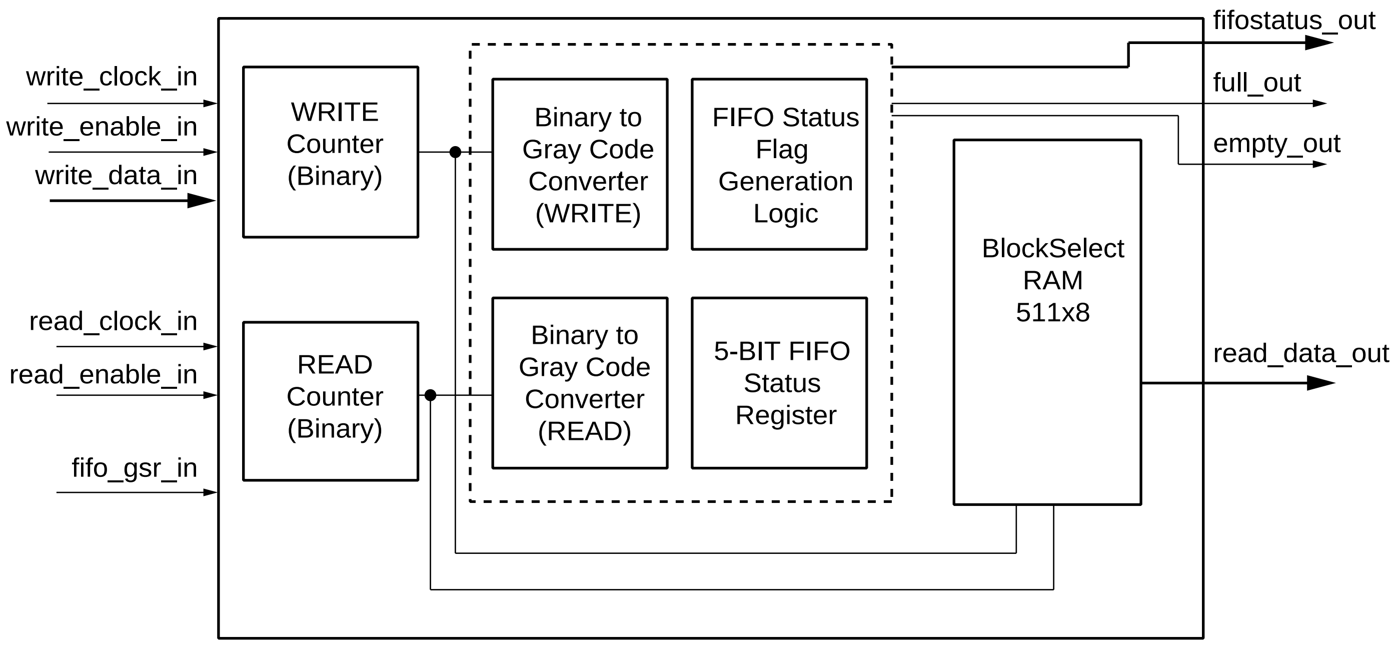
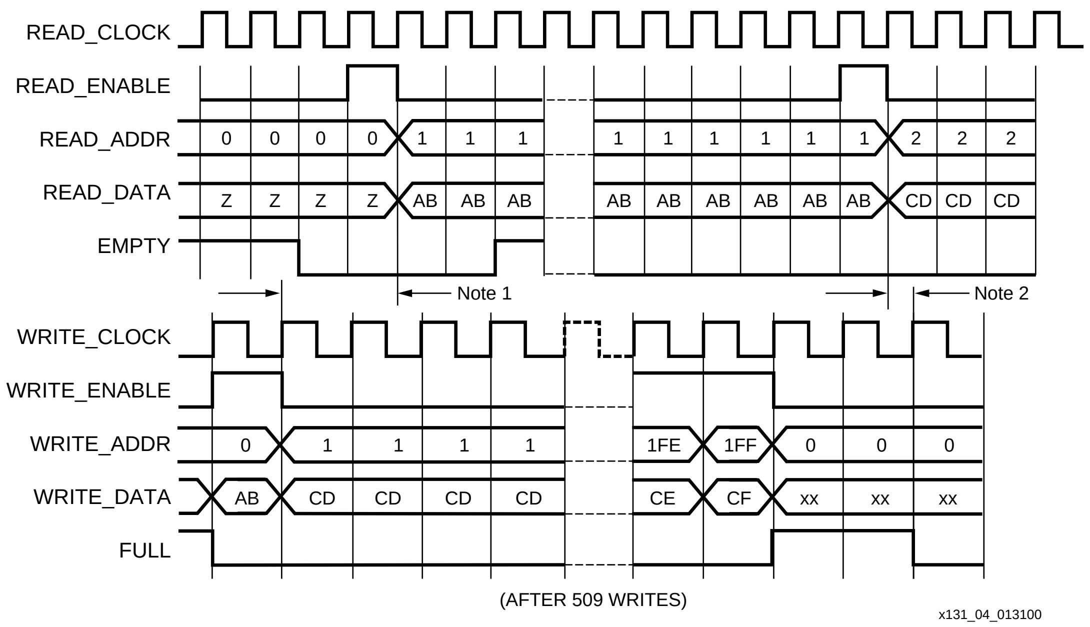


Figure 3: **511 x 8 Asynchronous FIFO**



Notes:

1. EMPTY will go Low if the write addresses meet the set-up 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 set-up time before the rising edge of the WRITE_CLOCK. If not, FULL will go Low one clock cycle later.

Figure 4: **511 x 8 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	8
read_enable_in	input	1
read_data_out	output	8
full_out	output	1
empty_out	output	1
fifostatus_out	output	5

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 words stored. Additional comparators are used 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).

Because, different from the common-clock version, it is not possible to keep a reliable count of the number of words in the FIFO, a FIFO status output is used instead. It is 5-bits wide, with the signals representing various ranges of fullness, as seen in Table 3.

Table 3: **FIFO Status Output Description**

FIFO Status Bit	Description
fifostatus[0]	FIFO is between Empty and 1/4 Full
fifostatus[1]	FIFO is between 1 word and 1/2 Full
fifostatus[2]	FIFO is between 1/4 full and 3/4 Full
fifostatus[3]	FIFO is between 1/2 Full and Full
fifostatus[4]	FIFO is between 3/4 Full and Full

The fifostatus outputs are mutually exclusive, meaning only one will be High at any one time, but the ranges that they cover overlap. They are based on the Gray-code pointers, and the quadrant deltas that exist between the Read and Write addresses. Because of the nature of Gray-code counting, more precision (such as one based on octants) can be easily added, because the upper bits of a Gray-code address are themselves Gray-coded, so there will not be any incorrect status registered.

The overall worst-case path is the generation of Full and Empty, and the delays for these paths are reported in the following table. Delays shown for hand-placed versions are near-optimal but not necessarily absolutely optimal.

A better implementation of the fifostatus can be found in **XAPP205**. The new design produces a binary count of the words in the FIFO, with some latency in generating the result. It gives an accurate count of the number of words in the FIFO.

Conclusion

The block SelectRAM can be used to generated both synchronous and asynchronous FIFOs in Virtex devices. Asynchronous FIFOs are possible due to the true dual-port nature of the block SelectRAM feature. These FIFOs can operate at speeds faster than 150 MHz. Some performance results are listed in Table 4. in the Virtex devices.

Table 4: **Performance Results**

Design Version	**Speed Grade**	
	-6	**-5**
Independent Clocks (automatic placement)	6.6 ns (150 MHz)	7.6 ns (130 MHz)
Independent Clocks (hand-placed)	5.9 ns (170 MHz)	6.7 ns (150 MHz)
Common Clocks (automatic placement)	6.0 ns (167 MHz)	6.9 ns (145 MHz)

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 in both VHDL and Verilog (File: **xapp131h.zip**). The design for Independent Read and Write clocks is fifoctlr_ic.v, and the design for Common Read and Write clocks is fifoctlr_cc.v.

Revision History

Date	Version	Revision
12/10/98	1.1	Updated from 1.0 for Applinx printing
9/22/99	1.2	Updated to include Virtex-E family
2/2/00	1.3	Reformatted document and added block and timing diagrams.