

## Summary

Two FIFO designs are described. In both cases, arbitration permits any RAM cycle to be a PUSH or a POP. XC4000 RAM performance is improved through read-modify-write operation, and the fastest clock required is at the RAM-cycle rate. The first design is expandable to any size FIFO, while the second, faster design is restricted to 16 or 32 words.

## Specifications

Maximum Clock Frequency (estimated for XC4000-5)  
16 x 8-bit FIFO 40 MHz

## LCA Family

XC4000

## Demonstrates

RAM Operation

## Introduction

The four components of a RAM-based FIFO are shown in Figure 1. To control the RAM, the address-logic block maintains two addresses, one for the current write location, where data is PUSHed, and one for the current read location, from which data is POPped. Following a PUSH or a POP, the corresponding address is incremented, causing data to be written and read sequentially.

The flag logic uses the read and write addresses to determine the status of the memory. If the memory contains no valid data, an EMPTY flag is created; a FULL flag is created when all memory locations are occupied.

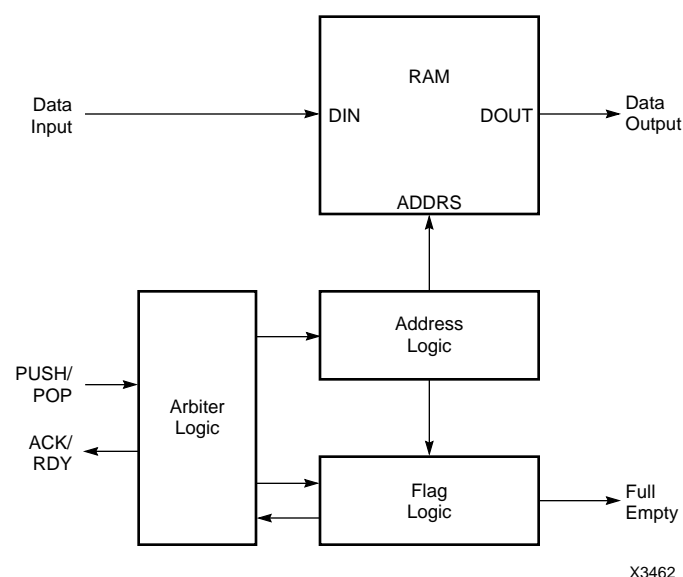


Figure 1. FIFO Block Diagram

The arbitration logic, for the most part, simply passes PUSH and POP requests to the RAM. Simultaneous PUSH and POP requests, however, must be resolved. The simplest schemes have a fixed priority, with either PUSH or POP being designated as the priority operation. If the priority operation is not possible because the RAM is full or empty, the priority should be overridden. Other schemes can alternate in priority between PUSH and POP, or favor one operation while its request persists.

Both FIFOs described depend on read-modify-write operation of the RAM, with Write Enable asserted every cycle. In the first design, a data multiplexer permits "non-write" cycles by rewriting existing data into the RAM. The same multiplexer provides bank selection for RAM expansion, permitting any size FIFO.

In the second design, the data multiplexer is omitted for faster operation. As a consequence, however, bank selection is not possible, and the RAM depth is limited to one CLB. Without a data multiplexer, only new data can be written in the RAM. During non-POP cycles, new data is always written to the current write address. If PUSH is asserted, the data is valid and the write address is incremented before the next write. If PUSH is not asserted, the address is not incremented, and the invalid data is subsequently overwritten. During a POP cycle, data read from the RAM is stored in a register, and the RAM location immediately overwritten with invalid data.

## Operating Description

### Expandable FIFO

Figure 2 shows the RAM and its address counters. A key element is the use of a BUFGS to drive the RAM Write Enable and clock the address register. The low skew of the global net ensures that the data and address hold



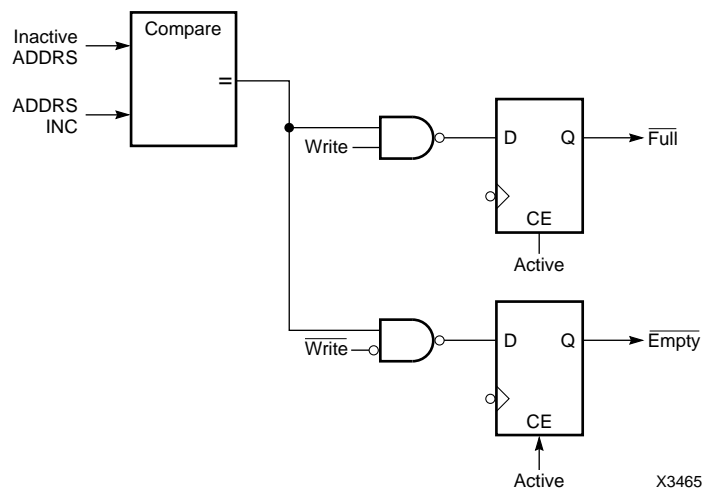


Figure 3. Flag Logic

cycle. If both are asserted together, the next operation in the next RAM cycle is determined by a user-defined Priority signal. Several options for the Priority signal are discussed later.

If a PUSH or a POP is requested and the FIFO is not full or empty, respectively, the next cycle is declared Active. A write or a read occurs and the corresponding address is incremented. Otherwise, the cycle is inactive; no read or write occurs and the addresses remain unchanged. In an inactive cycle, the Write signal is de-asserted by default.

Two handshake signals are generated. ACK acknowledges that a PUSH request will be honored in the next RAM cycle. Input data is captured on the falling clock

edge that starts the RAM cycle. RDY indicates that a POP request will be honored during the next RAM cycle. Output data is made available on the falling clock edge that ends the RAM cycle.

In deciding the next operation when both PUSH and POP are asserted, the most straightforward Priority functions simply default to one operation or the other. To always write, a logic High could be used, and to always read, a logic Low. In practice, however, this can lead to wasted cycle. For example, PUSH could win when the FIFO is full and the operation cannot be performed. A better choice is to use FULL as Priority to always select write unless the FIFO is full. Similarly, using EMPTY will cause POP to always win unless the FIFO is empty.

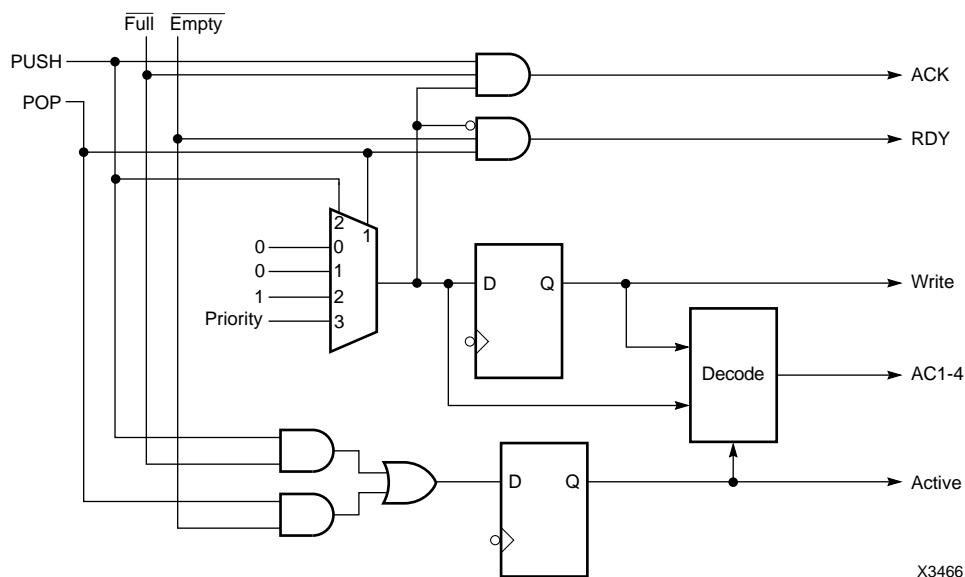


Figure 4. PUSH/POP Arbitration Logic

The above priorities are useful when receiving data from a burst source, such as a bus, or transmitting burst data. While burst is in progress, however, the other operation can be locked out for many cycles. If a more even resource allocation is required, Write can be used as Priority. In this case, requesting PUSH and POP continuously results in alternating reads and writes.

While the time to acquire the FIFO is reduced to no more than one cycle, the guaranteed peak PUSH/ POP rate is also reduced. In the limit, PUSH and POP may only operate at half the RAM cycle rate. The average data through the FIFO is unaffected, however. In the long term, it is obvious that no more than half the RAM cycles can be PUSHes. Attempting to achieve more will fail when the FIFO becomes full. Similarly, no more than half the cycles can be POPs, since the FIFO will become empty.

A third option permits both burst reads and burst writes, although either PUSH or POP may experience a long delay acquiring the FIFO if it is busy. Priority is connected to Write. As a result, the FIFO repeats its last operation whenever there is a conflict. A burst read or write will continue, and lock out the other operation until the burst is complete.

### High-Performance FIFO

The RAM block for the high-performance FIFO is shown in Figure 5. In this design, the RAM has simple input and output registers. The input register captures data on the falling edge of the clock which, in addition, marks the start of a RAM cycle. Data is captured in the output register at the end of the read phase, when the clock goes high.

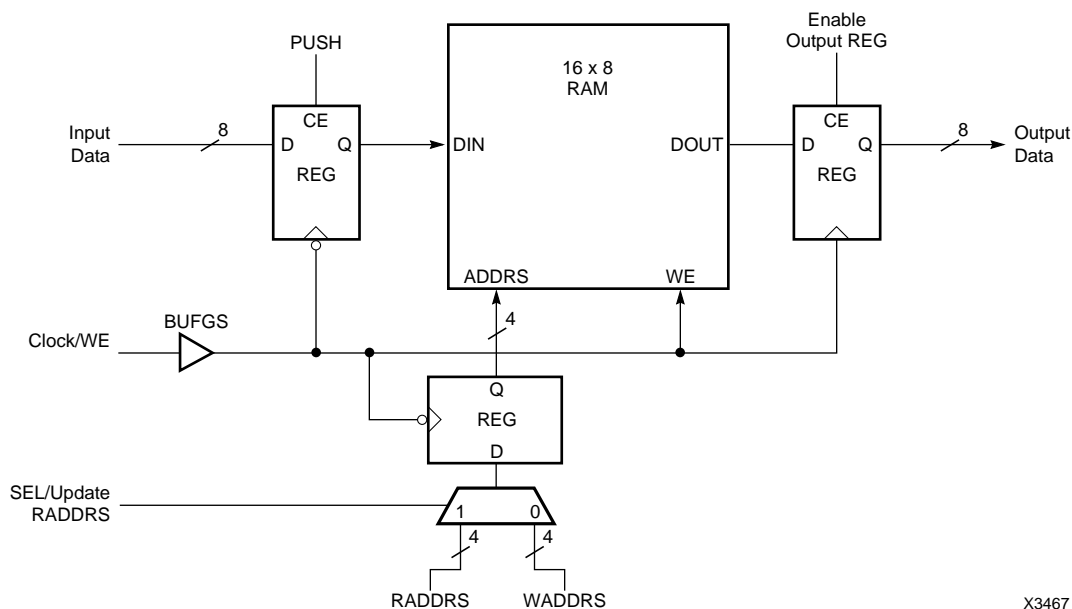
Prior to the start of the RAM cycle, a selection is made between the read and write addresses, and the selected address is registered when the RAM cycle starts. The read address is only selected when a POP is to be executed. Stable output data is retained from POP to POP, however, since the output register is only enabled during POPs. If the output data is required to change on the falling edge, an additional register must be used.

Since new data is written into the RAM every cycle, the read address cannot be selected during idle cycles; valid data waiting to be read would be destroyed by the write. Consequently, the write address is selected for both PUSH and idle cycles.

In the previous design, the RAM can be filled completely. When the FIFO is full, the next write address becomes equal to the next read address. This is not a problem, provided the location is read before it is overwritten. In the current design, there must always remain at least one unused location where invalid data is written during idle cycles.

When a PUSH occurs, the data written finally becomes valid, and the write address is incremented. During subsequent idle cycles, invalid data is written to the new write address. Overwriting of this address continues until the next PUSH.

As a consequence, a maximum of 15 words can be stored in the RAM. The FIFO can, however, store two more words in the input and output registers. The total storage capacity is 17 words.



X3467

Figure 5. RAM Diagram

Figure 6 shows the address and flag logic for the FIFO. Modified 4-bit linear-feedback-shift-register (LFSR) counters are used. Conventionally, 4-bit LFSR counter use the XNOR of the last two shift-register bits as feedback the input. This results in a sequence that repeats every 15 clocks. The missing count, all-1s, can, however, be added to provide the count sequence shown in Table 2.

Normally, 1110 is followed by 0111, and 1111 would cause the counter to lock up. To include 1111 in the sequence, 111X is detected, and the shift-register input is inverted while this condition is met. Consequently, 1110 is followed by 1111 and the next count is 0111, since the input remains inverted. The remainder of the count sequence is unaffected.

A benefit of LFSR counters in this design is that adding one extra bit to the shift register permits access to two adjacent addresses, which, in turn, permits easy generation of the FULL flag. The current write address is available to the RAM, while the next write address is also available for comparison with the read address. When the next write address equals the read address the FIFO is full. The current write address is the sixteenth RAM location needed for invalid data writes during idle cycles. The FIFO is empty when the current read address becomes equal to the current write address, which is yet to be written with valid data.

Table 2. Adder-Counter Sequence

Shift Register Input	
0	0 0 0 0
1	0 0 0 0
1	1 0 0 0
1	1 1 0 0
1	1 1 1 0
1	1 1 1 1
0	1 1 1 1
1	0 1 1 1
1	1 0 1 1
1	1 1 0 1
0	1 1 0 1
0	0 1 1 0
0	0 0 1 1
1	0 0 1 1
0	1 0 0 0
1	0 1 0 0
0	1 0 1 0
0	1 0 1 0
0	0 1 0 1
0	0 0 1 0
0	0 0 0 1
	0 0 0 0
	1 0 0 0
	⋮

X5281

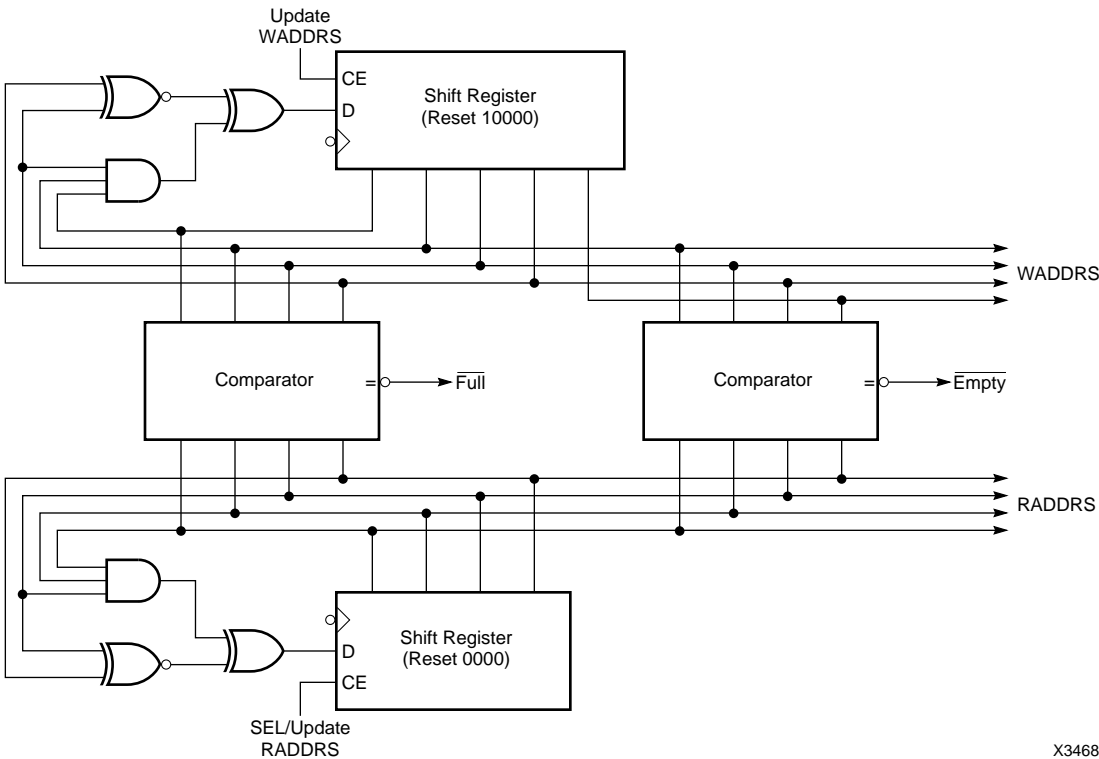
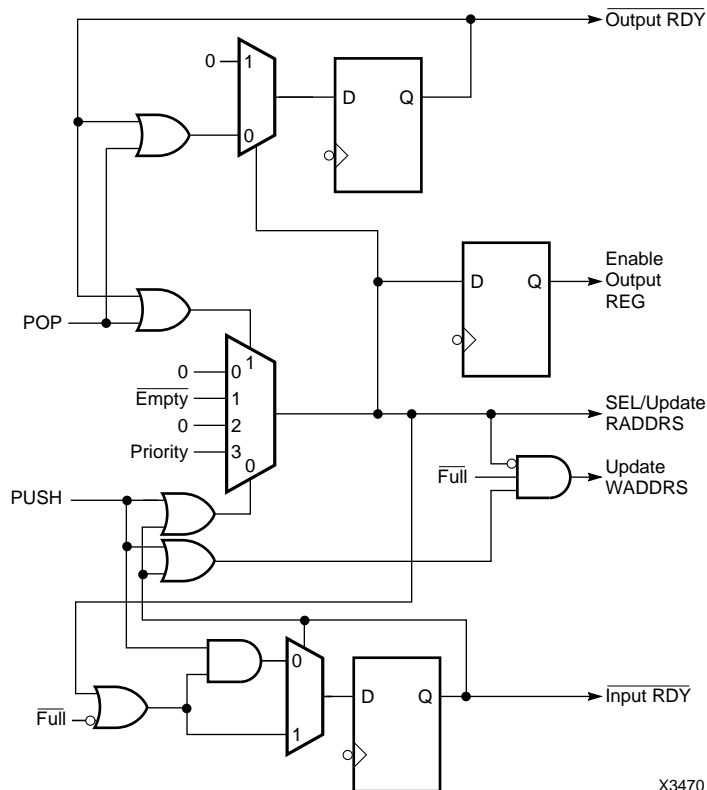


Figure 6. Address/Flag Logic



X3470

**Figure 7. PUSH/POP Arbitration**

The arbitration logic is shown in Figure 7. As in the previous design, the core of the arbiter is a multiplexer that selects the next operation according to the requests and the status of the RAM. The Priority input must be defined and implemented as discussed previously.

The output of the multiplexer is High when a read is to be performed. The Highfi causes the read address to be both selected for the RAM and simultaneously updated. Otherwise, the write address is selected. The write address is only updated, however, when a PUSH is requested and the UPDATE-WADDRS command is issued.

PUSH/POP requests and input data must set up to the falling edge of the clock, and POPed data becomes available on the subsequent rising edge. If a request cannot be serviced immediately, it is stored in one of two flip-flops, and a RDY is asserted on the falling clock edge at the start of the RAM cycle. If a request can be serviced, the corresponding RDY flag is never asserted.

When a PUSH is deferred, the input data is still captured in the input register, but it is not transferred to the RAM. In this case, RDY should suppress further PUSHes. The RDY flags are cleared at the start of the RAM cycle in which the request is serviced.