



# AdcBulkMem

**Marc Defossez**  
**Sr. Staff Applications Engineer**

Created: September 28, 2009  
Modified: December 8, 2009

© Copyright 2009 - 2009, Xilinx, Inc. All rights reserved.

# DISCLAIMER:

© Copyright 2009 - 2009, Xilinx, Inc. All rights reserved.

This file contains confidential and proprietary information of Xilinx, Inc. and is protected under U.S. and international copyright and other intellectual property laws.

## Disclaimer:

This disclaimer is not a license and does not grant any rights to the materials distributed herewith. Except as otherwise provided in a valid license issued to you by Xilinx, and to the maximum extent permitted by applicable law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND WITH ALL FAULTS, AND XILINX HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under or in connection with these materials, including for any direct, or any indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same.

## CRITICAL APPLICATIONS

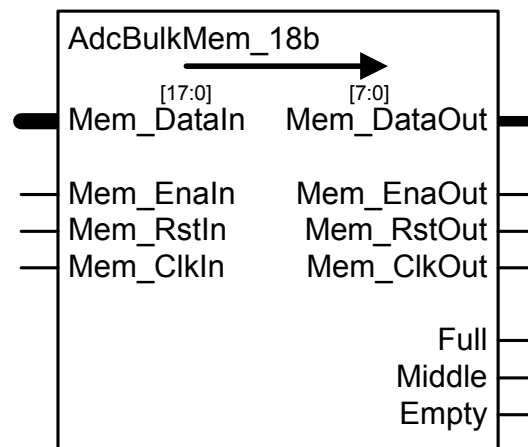
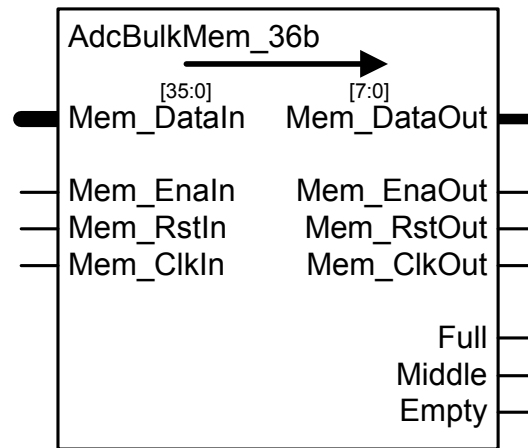
Xilinx products are not designed or intended to be fail-safe, or for use in any application requiring fail-safe performance, such as life-support or safety devices or systems, Class III medical devices, nuclear facilities, applications related to the deployment of airbags, or any other applications that could lead to death, personal injury, or severe property or environmental damage (individually and collectively, "Critical Applications"). Customer assumes the sole risk and liability of any use of Xilinx products in Critical Applications, subject only to applicable laws and regulations governing limitations on product liability.

THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS PART OF THIS FILE AT ALL TIMES.

Contact: e-mail [hotline@xilinx.com](mailto:hotline@xilinx.com) phone + 1 800 255 7778

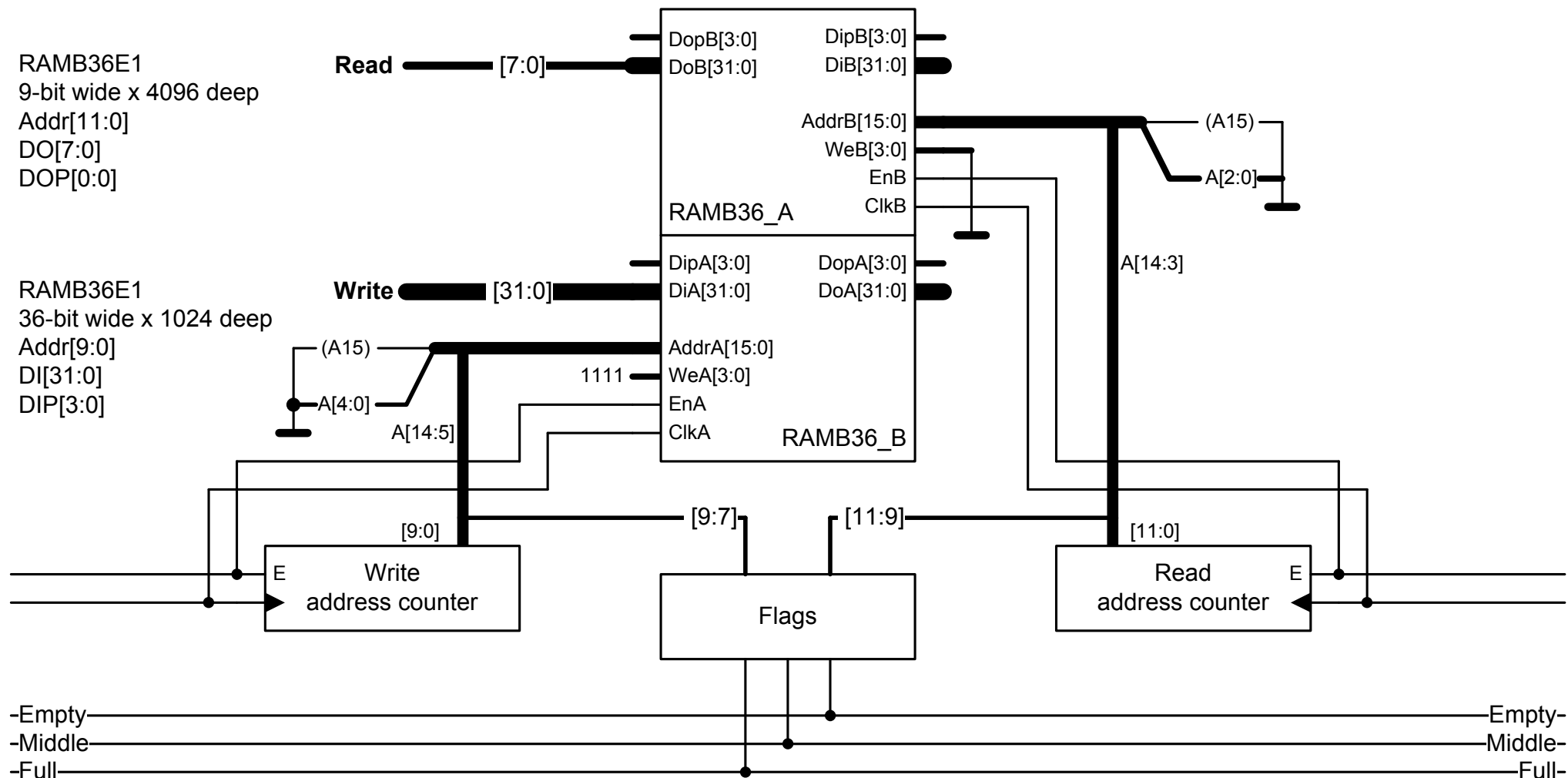
This page is intentionally left blank.

# AdcBulkMem

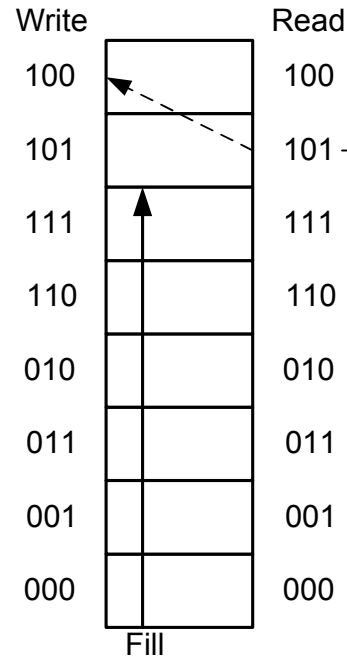


# AdcBulkMem

The ADC writes into the Block-RAM and the processor reads from it.



# Flags – Gray code



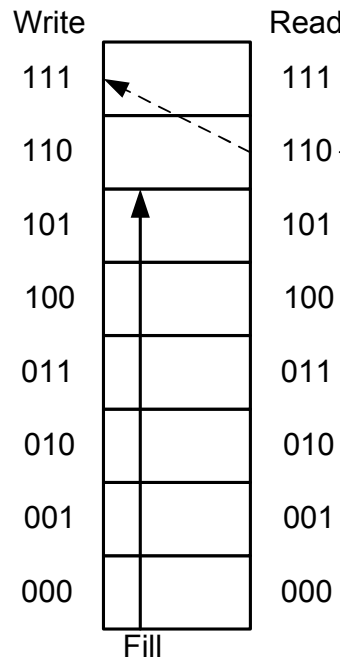
When the read pointer is right behind the write pointer, the FIFO is as good as empty.  
When the write pointer is right behind the read pointer, the FIFO is nearly full.

EMPTY		FULL		Optional MIDDLE	
100	101	100	000	100	010
101	111	101	100	101	011
111	110	111	101	111	001
110	010	110	111	110	000
010	011	010	110	010	100
011	001	011	010	011	101
001	000	001	011	001	111
000	100	000	001	000	110
Write	Read	Write	Read	Write	Read

Assume that the write pointer runs faster than the read pointer, then:  
The write pointer will reach “100” while the read pointer is still “000” the FULL flag goes high.  
Writing must now be stopped until the read pointer reaches “101”, now the EMPTY flag can be set high and the FULL flag can be released.  
Writing can continue from where it was stopped (Somewhere in the “100” region) and reading can be stopped. Writing goes over turn around in address space. When the write pointer reaches now “111 the FULL flag must be set again (Read pointer is at “101”). The read operation can start from where it last. When the read pointer reaches “110” the EMPTY flag can be set (reset the FULL flag).  
And so on.....

# Flags – Binary

When the counters are binary coded the flag logic should look like this;



When the read pointer is right behind the write pointer, the FIFO is as good as empty.  
When the write pointer is right behind the read pointer, the FIFO is nearly full.

EMPTY		FULL		Optional MIDDLE	
111	110	111	000	111	011
110	101	110	111	110	010
101	100	101	110	101	001
100	011	100	101	100	000
011	010	011	100	011	111
010	001	010	011	010	110
001	000	001	010	001	101
000	111	000	001	000	100
Write	Read	Write	Read	Write	Read

Assume that the write pointer runs faster than the read pointer, then:

The write pointer will reach “111” while the read pointer is still “000” the FULL flag goes high.

Writing must now be stopped until the read pointer reaches “110”, now the EMPTY flag can be set high and the FULL flag can be released.

Writing can continue from where it was stopped (Somewhere in the “111” region) and reading can be stopped. Writing goes over turn around in address space. When the write pointer reaches now “101 the FULL flag must be set again (Read pointer is at “110”). The read operation can start from where it last. When the read pointer reaches “100” the EMPTY flag can be set (reset the FULL flag).

And so on.....