BramDeSerializer

Marc Defossez
Sr. Staff Applications Engineer

Created: December 2, 2009
Modified: May 5, 2010

DISCLAIMER:

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

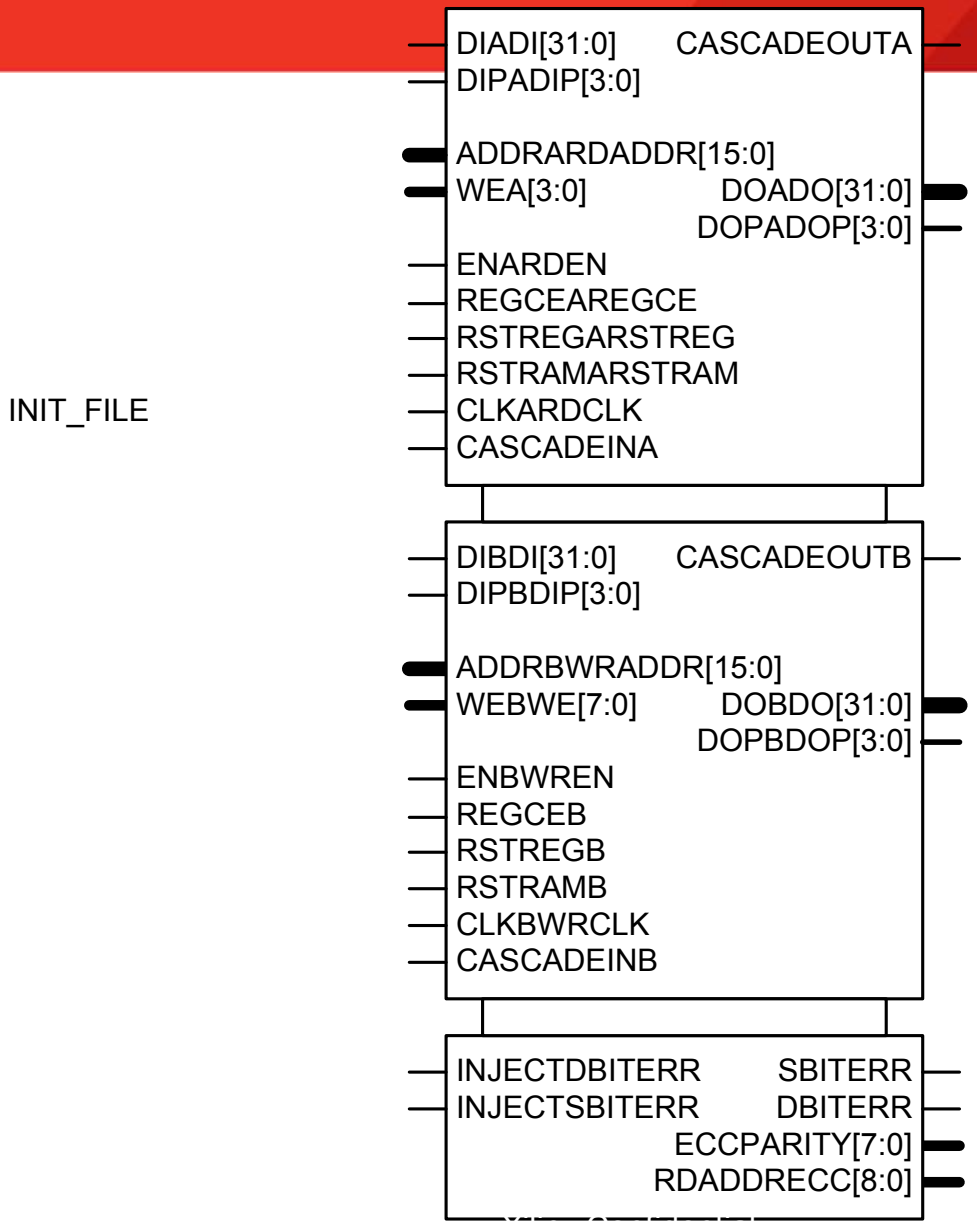
Xilinx products are not designed or intended to be fail-safe, or for use in any application requiring fail-safe performance, such as life-support or safety devices or systems, Class III medical devices, nuclear facilities, applications related to the deployment of airbags, or any other applications that 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 phone + 1 800 255 7778

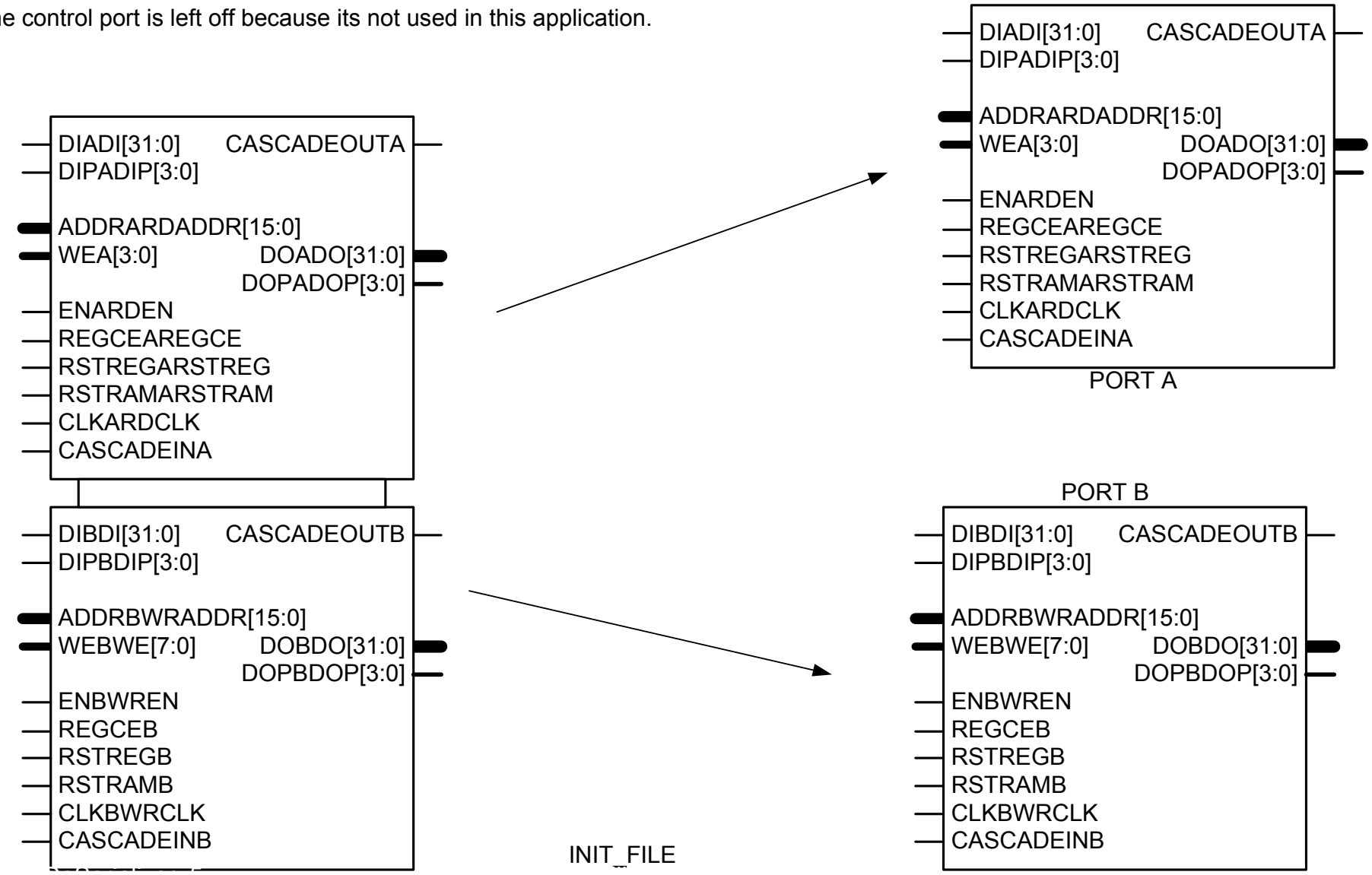
This page is intentionally left blank.

RAMB36E1



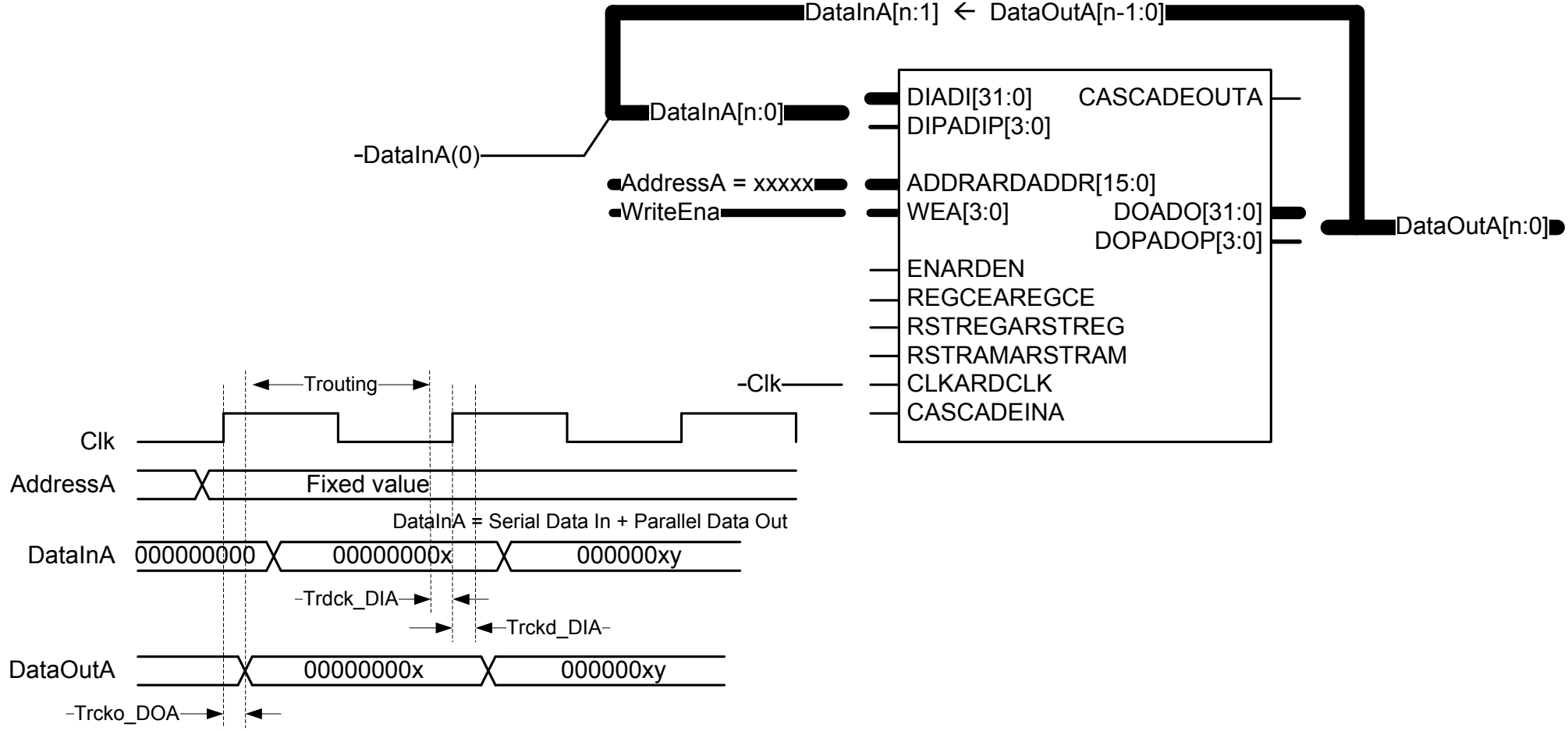
RAMB36E1 Used in this document

The control port is left off because its not used in this application.



DeSerializer

This sheet shows a 36-bit serial-to-parallel register. Any possible setup of the RAMB36E1 can be turned into a serial-to-parallel register.
This is a waste of a RAMB36 because only one address is used to make a serila-to-parallel register.
Hint: With less data bits needed it might be possible to include the address counter into the Block-RAM itself.

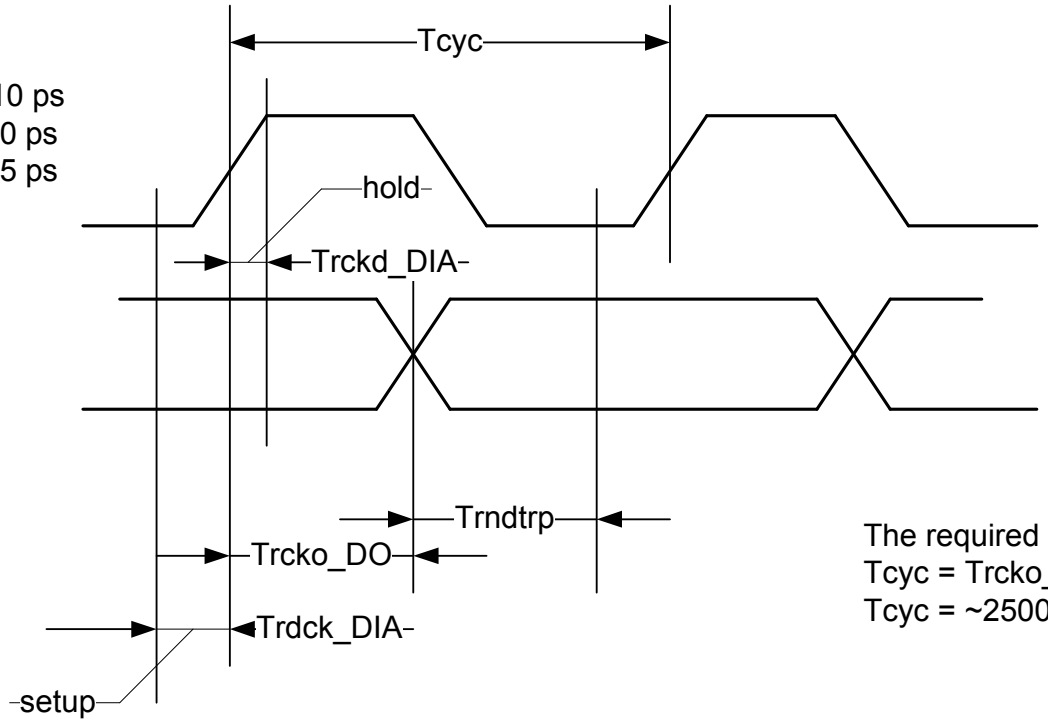


DeSerializer Timing

The most important timing for this application is the routing from the output of the Block-RAM back to the input of the Block-RAM. Let's call this timing Trndtrp (T_{Round_Trip}).
This routing must thus be placed under strict timing control when the highest possible speed must be obtained from the de-serializer.

The total round truip timing and thus the maximal operation frequency also depents on the parameters of the RAMB36E1 block. The figure shows a calculation example for a XC6VLX240T–2 component.

Average values:
Trndtrp = 1000 ps
Trcko_DO(DOP) = 1110 ps
Trdck_DIA(DIPA) = 420 ps
Trckd_DIA(DIPA) = 255 ps



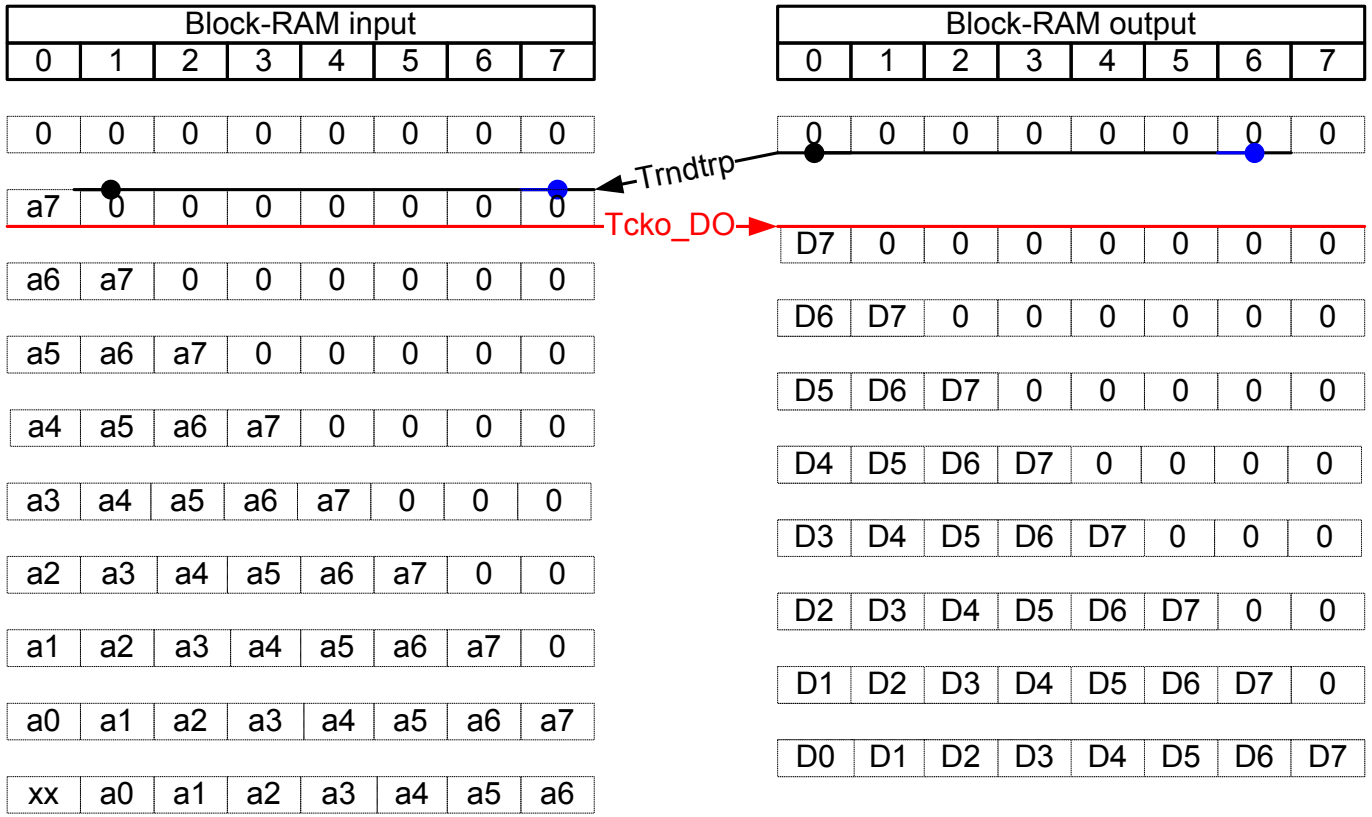
The required timing for the system is then:
Tcyc = Trcko_DO + Trndtrp + Trdck_DIA
Tcyc = ~2500 ps or 2.5 ns = 400 Mhz

DeSerializer Data Flow

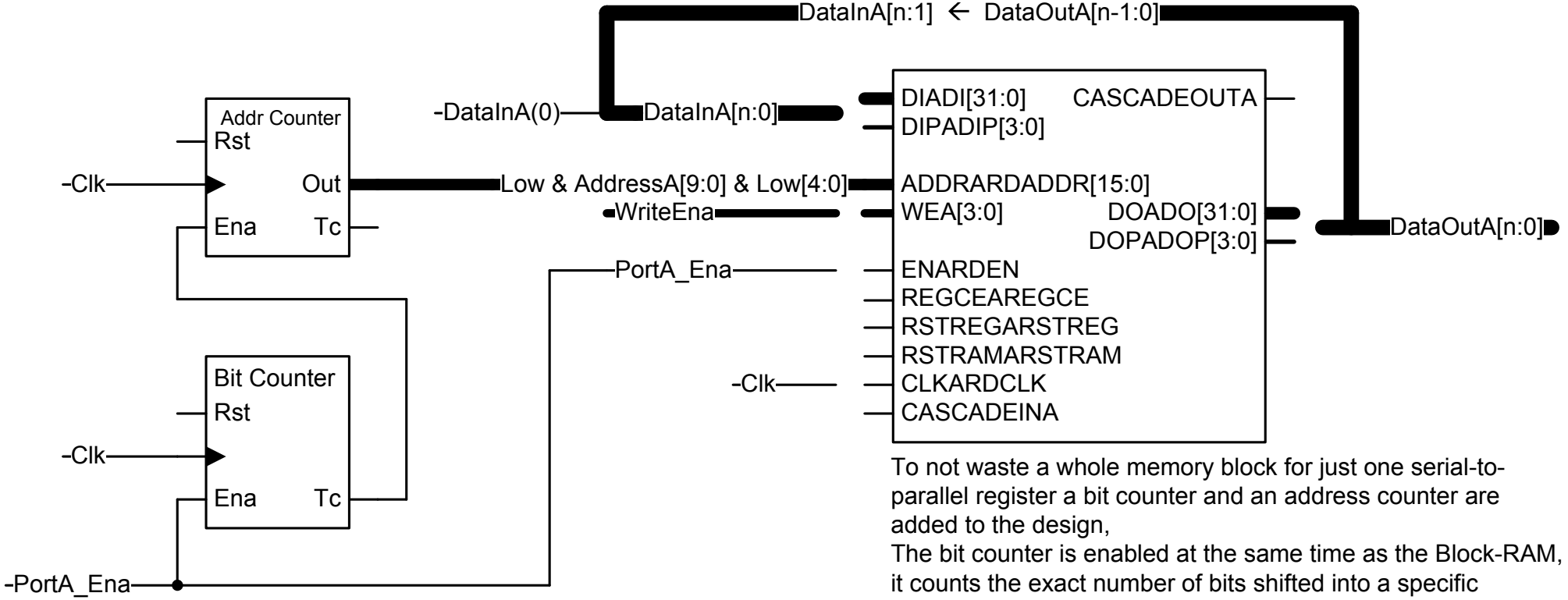
Just to show how the data flows in and out of the Block-Ram a eight bit example is given here

The serial input data is received MSB first and is named: a7, a6, down to a0.
The output data of the Block-RAM is named D7, D6, D5 down to D0.

Output bit 0 is routed back to input bit 1.
Output bit 1 is routed back to input bit 2
……
Output bit 6 is routed back to input bit 7



DeSerializer with a Twist

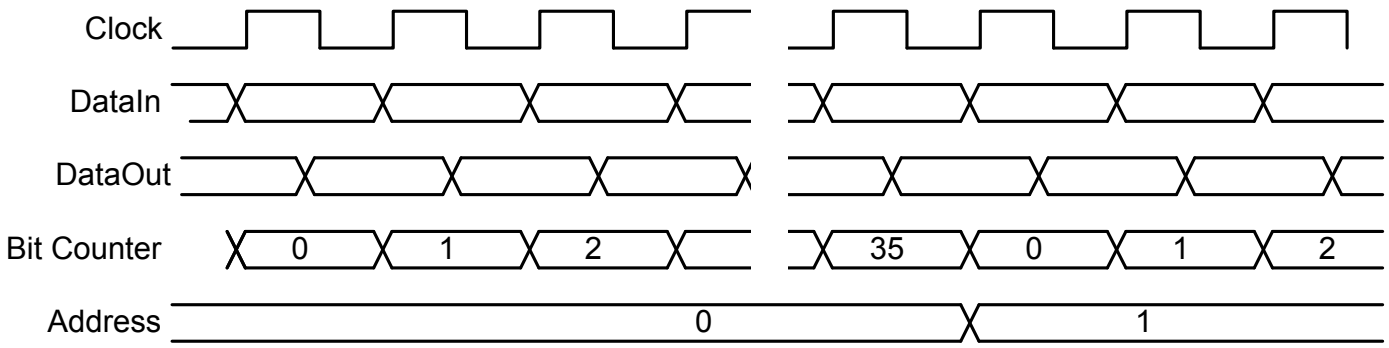


The "Bit counter" must count as far as the data depth of the Block-RAM. Example when the Block-RAM is set to 32-bit+4-bit and all 36-bit are used as shift register, the counter must count from 0 to 35 and issue a terminal count at bit 36.

When all address are written, Port A must be set in read mode in order to be able to read all memory contents via the data out bus.

To not waste a whole memory block for just one serial-to-parallel register a bit counter and an address counter are added to the design,
The bit counter is enabled at the same time as the Block-RAM, it counts the exact number of bits shifted into a specific memory address.
Except counting the written bits, so far nothing has changed since the previous example.
When the bit counter enables at it's terminal count an address counter to increase the address by one suddenly the design becomes interesting.
The Block-RAM is now changed into a pile of serial-to-parallel registers or otherwise written the Block-RAM becomes a serial-to-parallel register with a build in memory/FIFO.

DeSerializer with a Twist (waveforms)



When the bit counter has seen 36 new serial bit shifted into the Block-RAM then its terminal count becomes active and the address counter is enabled to point to the next address.

Before rising clock edge.
DataIn = New Serial input bit at DataIn(0) and looped back data from DataOut(n-1:0)

After rising clock edge.
DataOut = New Serial data and looped back data from DataOut(n-1:0)

DeSerializer with a Twist and a Turn

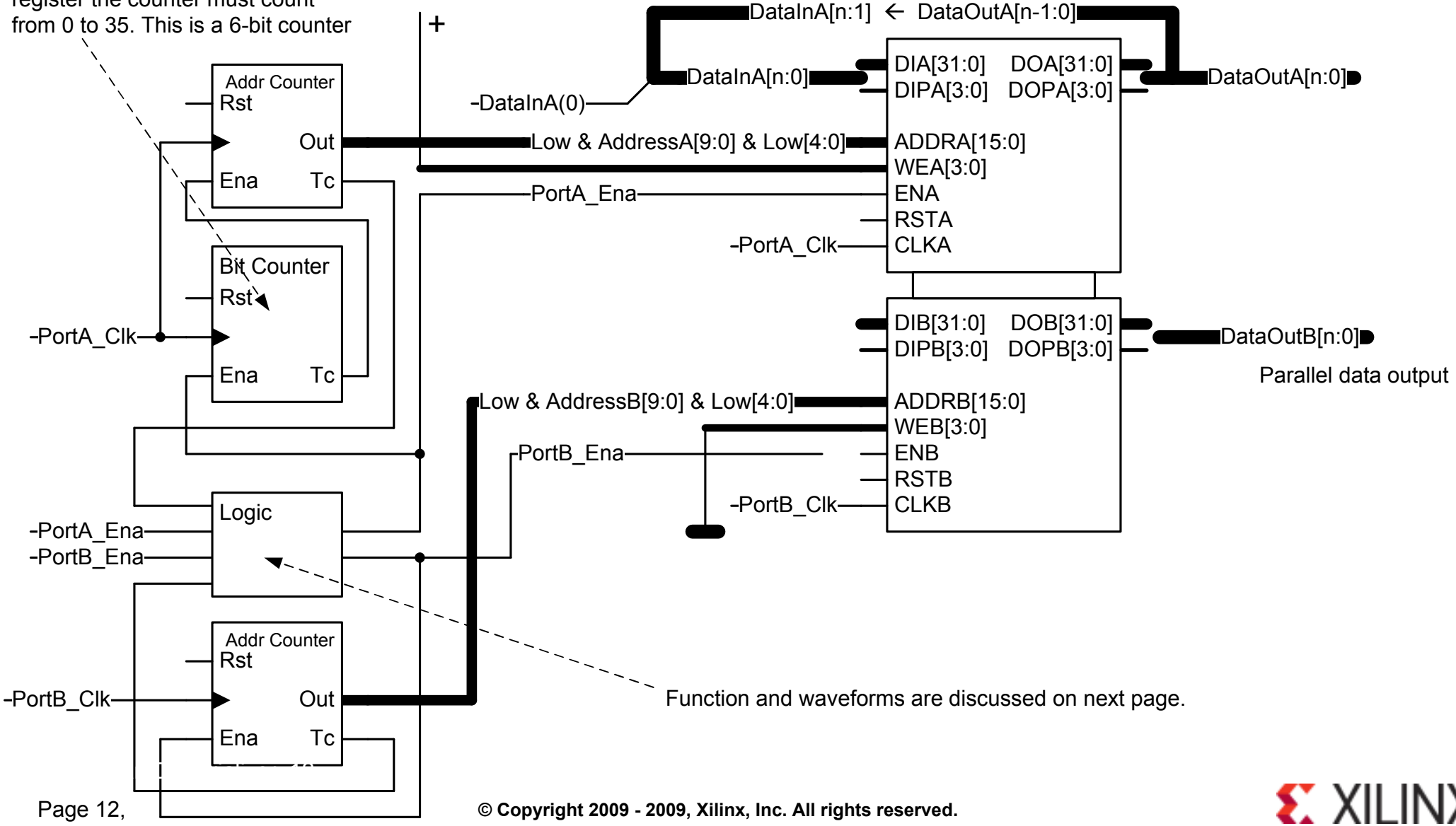
For the examples on following slides the Bloc-RAM is reduced to it's important, tor the application, pins.

The first example adds the dual port capability of the Block-RAM to the design. This way it becomes possible to serially shift data into the Block-RAM, fill the Block-RAM as memory and read the contents in parallel format from port B.
The memory functionality is transformed into a kind of FIFO.
The memory is filled via serial-to-parallel conversion and when a certain address or address range is reached the memory content must be read via port B.

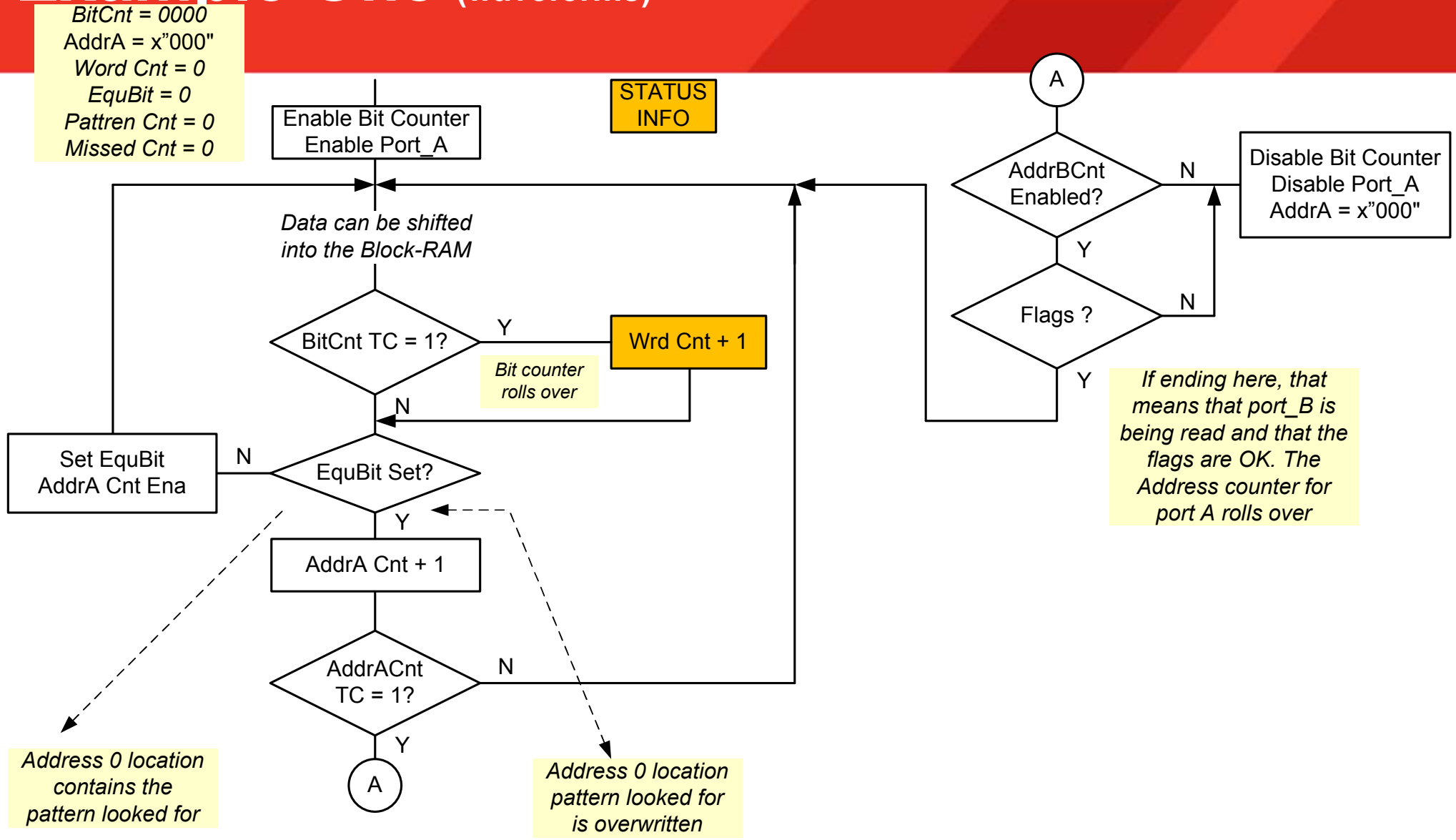
The second example adds something extra to the first example. A pattern comparator. This way it is possible to search a serial bit stream for a certain pattern before starting real reception of serial data. When the pattern is detected the address counter is incremented. Thus only good data is stored into the Block-RAM. As in the previous example it is read throught the B port of the Block-RAM

Example One

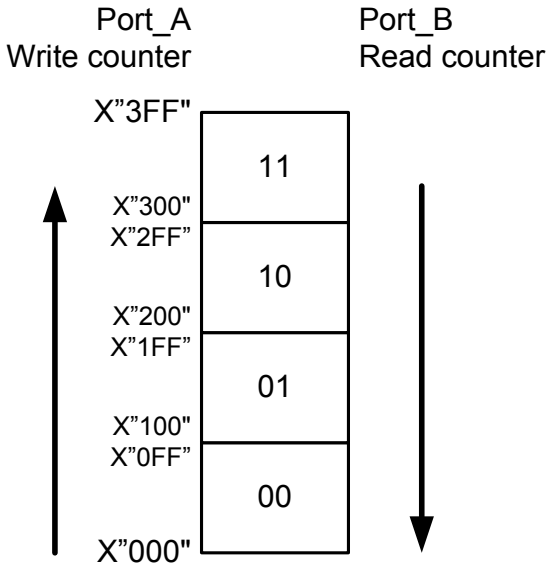
For a 36-bit serial-to-parallel register the counter must count from 0 to 35. This is a 6-bit counter



Example One (waveforms)



Example One (Block-RAM address counters & flags)

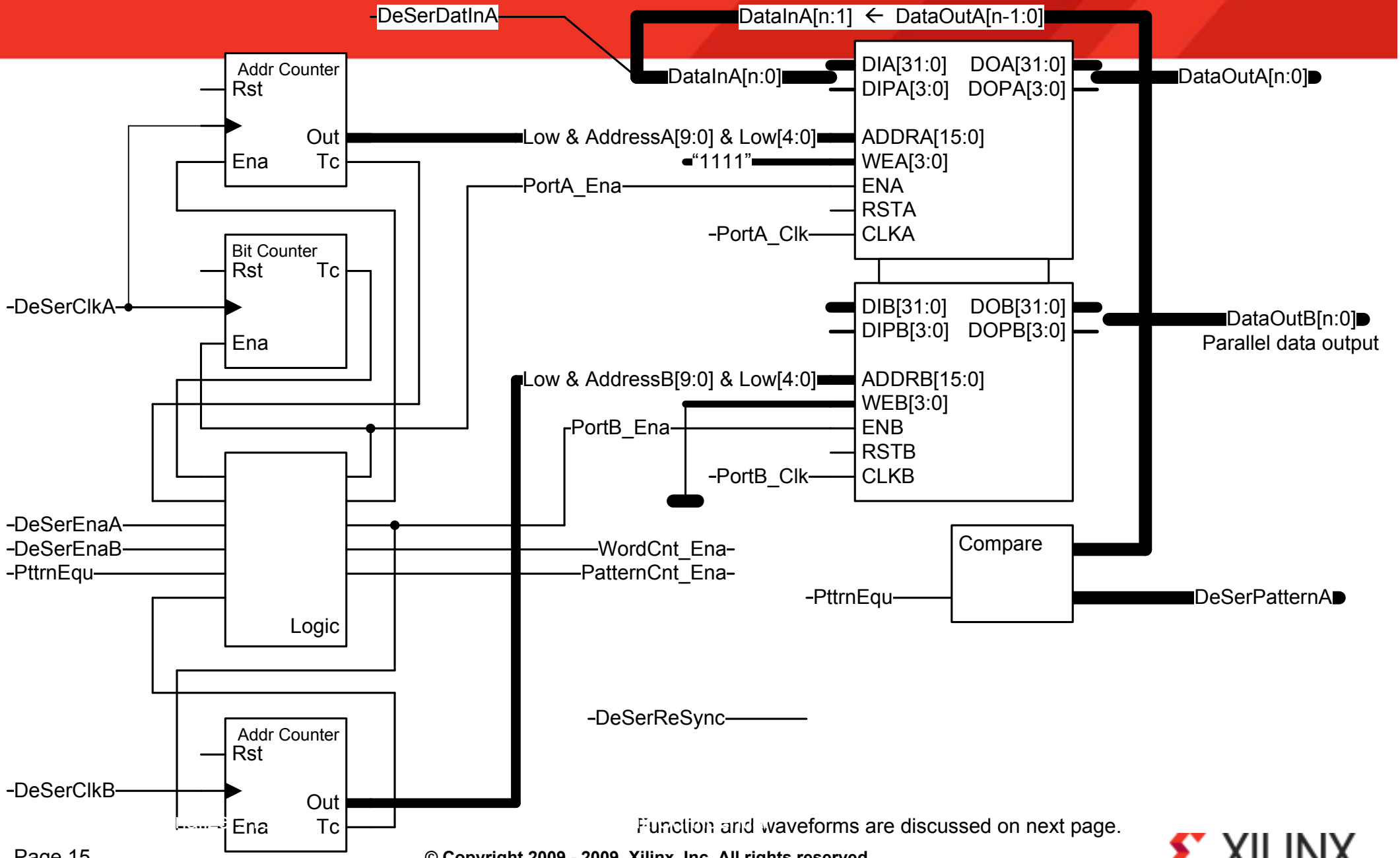


When the read pointer is right behind the write pointer, the FIFO is as good as empty.
When the write pointer is right behind the read pointer, the FIFO is nearly full.

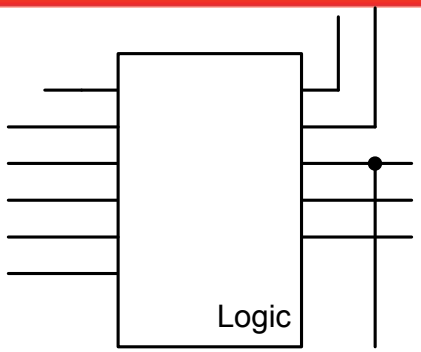
EMPTY			Lo-MID			Hi_MID			FULL		
11	=	11	00	=	11	01	=	11	10	=	11
10		10	11		10	00		10	01		10
01		01	10		01	11		01	00		01
00		00	01		00	10		00	11		00
Wr		Rd	Wr		Rd	Wr		Rd	Wr		Rd

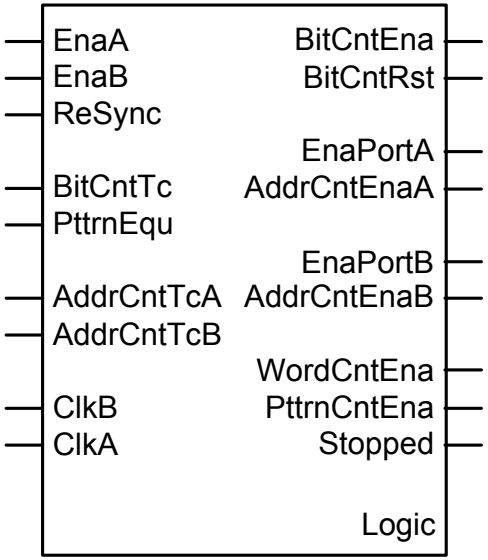
When both pointers are at the same counter value the buffer is considered to be empty, the flags indicate that the buffer is EMPTY.
Assume that one starts writing. Then the write pointer will shift one level up and the flags indicate that the buffer is filled up to the Lo-MID range. When the buffer continues to be written the flag will change to the Hi-MID indication and finally when no read operation is started the FULL flag will be raised.

Example Two



Function and waveforms are discussed on next page.





Example Two (waveforms)