GENERIC FIFO
Design tutorial
Technical Specification

Author: Tim Dan
tim.dan@vlsilab.com

Revision History

Revision	Author	Description	Date
1.0	Tim Dan	Initial	28.10.2007

Table of content

Contents

List of figures

List of tables

1. Introduction

1.1 Main Concept

The idea is to take a memory array (of any kind) and use it to implement a generic FIFO. Two address pointers are used to define the head and the tail of the FIFO. The queue will be implemented in a cyclic mode of operation, and the FIFO is defined to be full when the head pointer and the tail pointer are equal. An asynchronous logic system will control the two registers to containing head and tail addresses (write and read pointers).

The FIFO will be initiated using a reset signal, it will support 3 commands - write, read, pop and reset. The FIFO will send a signal if it is full (is_full output bit 1 is for full, 0 otherwise).

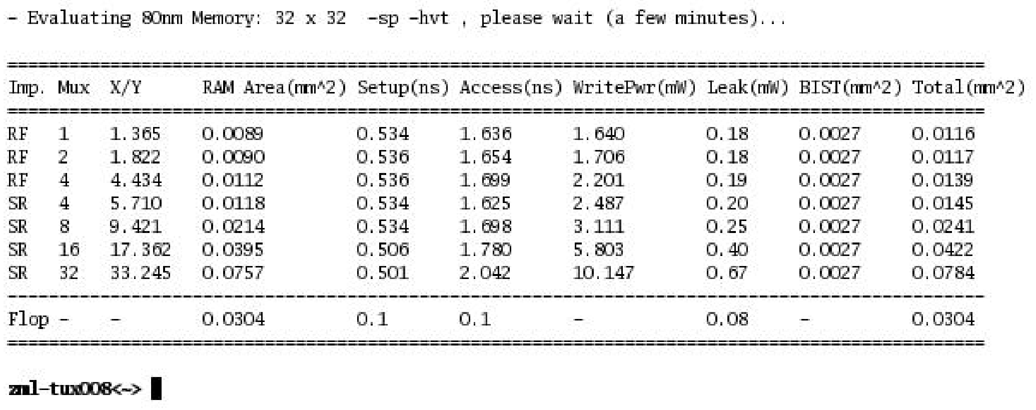
1.2 GENERAL INFORMATION and SIZE EVALUATION

The memory array will be automatically generated by a RAM generation program. The Width and Depth are supplied to the program which then generates a single cycle RAM memory array of the requested size.

In order to help the user meet the design specs in terms of timing, size and power, a memory evaluation script may be run on the different alternatives as follows :

USAGE: evaluate_ram_size.scr –depth [size in bits] –width [size in bits]

EXAMPLE of the ram evaluation tool output on a 32X32 bit memory array



```
- Evaluating 80nm Memory: 32 x 32  -sp -hvt , please wait (a few minutes)...

==========================================================================================
Imp. Mux  X/Y      RAM Area(mm^2) Setup(ns) Access(ns) WritePwr(mW) Leak(mW) BIST(mm^2) Total(mm^2)
==========================================================================================
RF    1   1.365    0.0089         0.534     1.636      1.640        0.18     0.0027     0.0116
RF    2   1.822    0.0090         0.536     1.654      1.706        0.18     0.0027     0.0117
RF    4   4.434    0.0112         0.536     1.699      2.201        0.19     0.0027     0.0139
SR    4   5.710    0.0118         0.534     1.625      2.487        0.20     0.0027     0.0145
SR    8   9.421    0.0214         0.534     1.698      3.111        0.25     0.0027     0.0241
SR    16  17.362   0.0395         0.506     1.780      5.803        0.40     0.0027     0.0422
SR    32  33.245   0.0757         0.501     2.042      10.147       0.67     0.0027     0.0784
------------------------------------------------------------------------------------------
Flop -    -        0.0304         0.1       0.1        -            0.08     -          0.0304
==========================================================================================

zml-tux008<-> █
```

The user may use the timing (setup, hold times), size and power data to enable him to make a better decision with regards to the best RAM configuration for the FIFO memory.

1.3 COMMANDS DESCRIPITON

RESET – (00 in the command line 2bit) empties the queue . Note that this command does not really delete the data , it simply changes the relevant pointers , so this process does not take more than one cycle. The changes are made at the beginning of the next cycle and therefore the input should be stable throughout the whole cycle.

READ - (01 in the command line 2bit) reads the head data line and outputs it by the output port. THIS COMMAND DELETES THE HEAD LINE DATA AND POPS IT.
Assumptions: the data will be stable not at the very beginning of the cycle but only after Logic TPD + access time. It is possible to determine the exact timing after choosing the specific memory type implementation.

WRITE – (10 in the command line 2bit) stores the input vector in the bottom of the FIFO unless the is_full signal is on.
Assumptions: the write operation complete at the beginning of the following cycle. Therefore the input vector as well as the command line should be stable for the whole cycle.
(NOTE – this is the worst case scenario – the time for the data to be stable can be reduced if the specific setup time is known.

1.4 PARAMETERS

The FIFO accepts 3 parameters

1. WIDTH – line size.
2. DEPTH – depth size.
3. LOGDEPTH – log2(DEPTH).

NOTICE:
Incorrect definition of parameters, especially DEPTH and LOGDEPTH will cause bugs.

1.5 THROUGHPUT/LATENCY

Throughput – will be determined by considering the read/write time of memory array.
Latency – two steps machine = as for time: 2 * throughput time.

2. Overview

2.1 GENERIC FIFO : Top Level Architecture

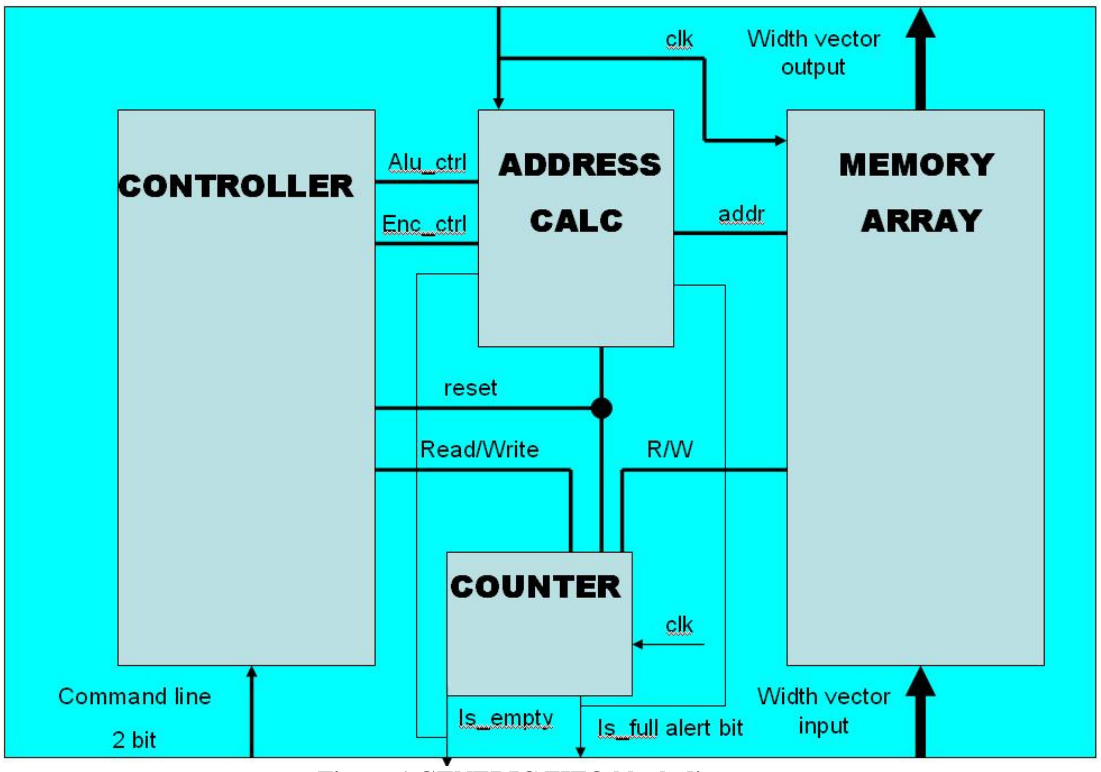


Figure 1 GENERIC FIFO block diagram

Briefly describe the high level functionality of the block (data path, processing and control path). Add block diagram, clocking scheme, and synchronization related issues.

2.2 GENERIC FIFO interfaces

Name	Width	I/O	Description
clk	1	I	Main clock @ 108MHz.
Command_line	2	I	Command from user 00 – reset, 01 – pop, 10 – write, 11 – read
Width_Vector_input	Width	I	Width bits input vector
Width_Vector_output	Width	O	Width bits output vector

Is_full	1	O	1 – for full queue , 0 - otherwise
Is_empty	1	O	1- for empty queue 0 - otherwise

Table 1 FIFO interface

Add a waveform

Figure 1 MyBlock->NextBlock write transaction

3. GENERIC FIFO Sub-bocks

3.1 CONTROLLER

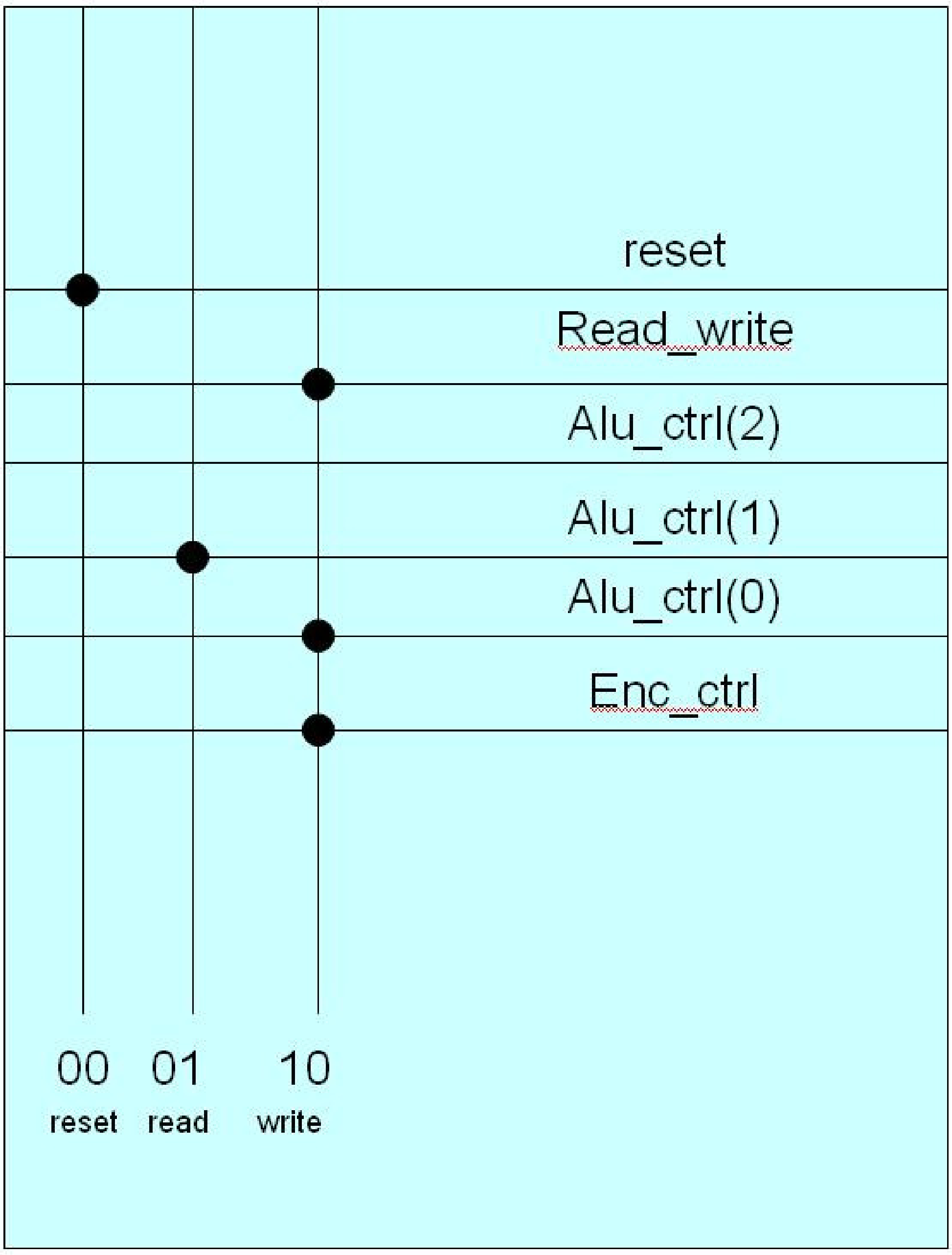
3.1.1 Functional description

The controller is a logic unit that takes the command requested and sends the appropriate control signals to the other sub blocks.

3.1.2 Interface

Name	Width	I/O	Description
Command line	2	I	Command line . 00 reset. 01 pop . 10 write. 11 read.
reset	1	O	Resets the registers when equals 1.
Read_write	1	O	Read or write command to counter
Alu_ctrl	3	O	A command line (3 bit) which Specify which operation to execute: 000 – outputs head. 001 – Increase head by 1. 010 – Decrease head by 1. 011 – Increase tail by 1. 100 – Decrease tail by 1.
Enc_ctrl	1	O	Pop = 0 read = 1

3.1.3 Implementation



3.2 ADDRESS CALCULATOR

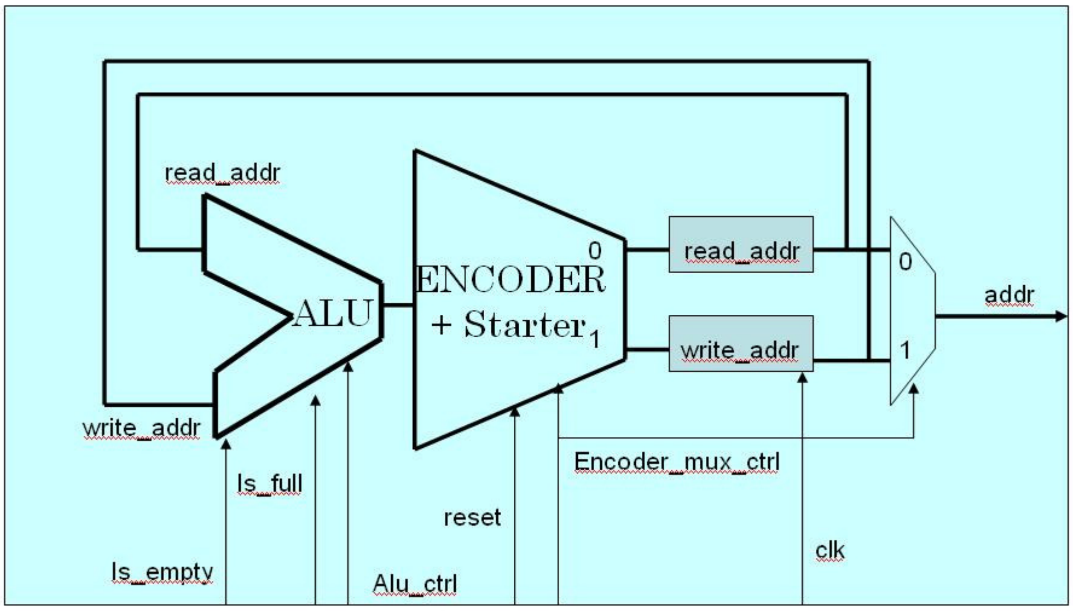


Figure 3 ADRESS CALCULATOR block diagram

3.2.1 Functional description

The address calculator manages the pointers to the top and the bottom of the queue including updating them when necessary.
ASUMPTIONS: User will not try to write when Is_full is true ("1") and will not try to read when is_empty is true ("1").

IMPORTANT: It is necessary to reset the unit to initialize its operation in order to initialize the address pointers shown in Fig 3.

3.2.2 Functional Units

The Address Calc is composed of the following subunits:

SubUnits	Functional description
ALU	Receives an input of 2 addresses of log[width] of head and tail, as well 3 bits for the Alu_ctrl. It outputs the next address as requested, plus an (is_full) indicator.
ENCODER+STARTER	It receives a reset signal and it initializes the head address as 1 and tail address as 0, otherwise it outputs the Alu output to the requested register.
REGISTERS (head_addr and tail_addr)	The size of the registers is determined by LOG[Width] , and each keeps the address of the current head and tail addresses.
Mux 2 to 1	Receives as inputs the outputs of the two registers and outputs

	the requested address (which determined by the ctrl bit).

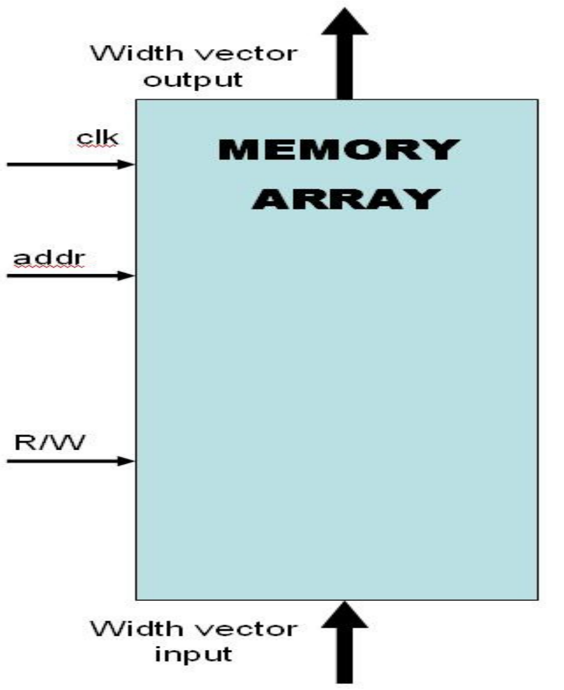
3.2.3 I/O Interface

Name	Width	I/O	Description
clk	1	I	Main clock @ 108MHz.
reset	1	I	Resets the registers when equals 1.
Encoder_mux_ctrl	1	I	Specifies the current address to flow
Alu_ctrl	3	I	A 3 bit command line which Specify which operation to execute: 000 – outputs head. 001 – Increase head by 1. 010 – Decrease head by 1. 011 – Increase tail by 1. 100 – Decrease tail by 1.
Is_full	1	I	1 – for full queue , 0 - otherwise
Is_empty	1	I	1 – for empty queue , 0 - otherwise
addr	LOG[width]	O	Requested address

3.3 MEMORY ARRAY

3.3.1 Functional description

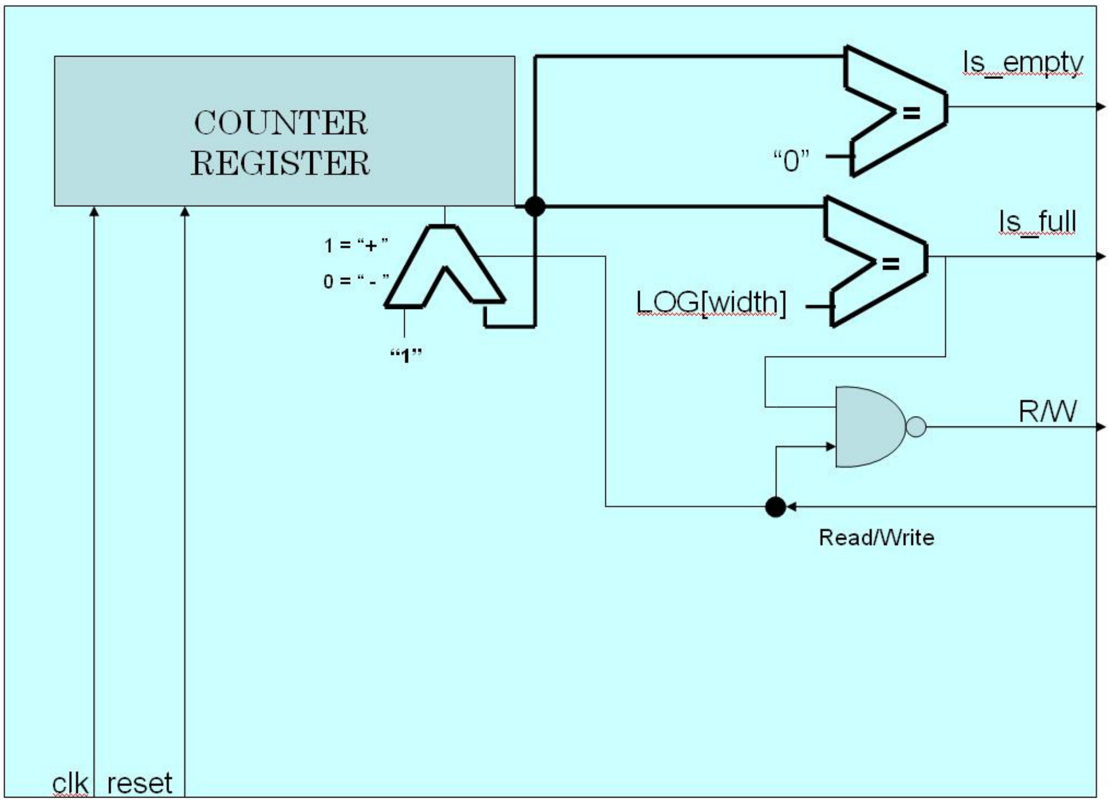
The Memory block is defined by the user according to his needs with regards to optimization considerations using evaluate_ram_size tool.



3.3.2 IO Interface

Name	Width	I/O	Description
clk	1	I	Main clock @ 108MHz.
Width vector output	Width	O	Output Vector.
Width vector input	Width	I	Input Vector
R/W	1	I	Read = 0 write = 1
addr	LOG[width]	I	Address of requested line

3.4 COUNTER



3.4.1 Functional description

The purpose of the counter block is to count the number of elements stored in the queue and to keep track of changes. It may be reset when necessary.

3.4.2 Counter : Sub-Blocks

element	Functional description
Counter register	Register Size : LOG[Width]. Counts number of elements in queue
Small Alu	Add or subtract left foot from right foot , with the read/write ctrl . If read (0) then it subtracts unless pop_read = 1 (read) and then it adds 0 . if write (1) it adds.
Equalizers	1 if equal , 0 otherwise

3.4.3 IO Interface

Name	Width	I/O	Description
clk	1	I	Main clock @ 108MHz.

reset	1	I	Resets the registers when equals 1.
Is_full	1	O	1 – for full queue , 0 - otherwise
Is_empty	1	O	1 – for empty queue , 0 - otherwise
R/W	1	O	Outputs read/write command to the memory array

4. Verification

- Propose verification test plan
- Critical points to cover
- Refer to points to cover
- Define verification techniques.
-

5. Synthesis and Layout

- RAM list
- Area for each block.
- Constraints for the tools
- Synthesis and layout results

6. MyBlock power control (if part of the design)

- Prove that your block consumes the lowest power it needs.
- Define power modes, and add description for each mode.

7. Appendix

- Put here all what you want to say, but you did not find a suitable chapter for it.
- For example ….

a. Useful path

- Path to top level RTL and internal modules.
- Path to digital IPs. Add version details.
- Path to analog IPs library.
- Path to I/O related library.

i. Verification

- Path to test base.
- Path to latest run results.

ii. Synthesis

- Path to latest synthesis results
- Path to special PT scripts.

iii. Power

- Path to latest run of power estimation tool.

b. Back End guidelines

- Special constrains and motivation for synthesis / P&R tools.
- Special scripts description for STA checking.