# *MEMORYmaster™ User's Guide*

**PRE-PRODUCTION**
**For the latest version of this document, check the User area on Actel's website at:**

**http://www.actel.com/user**

*WindowsNT™* and UNIX *Environments*

**Actel Corporation, Sunnyvale, CA 94086**


Printed in the United States of America

Part Number: 5579017

Release: June 1999



Trademarks

# Table of Contents

# List of Figures

# List of Tables

# Introduction

The *MEMORYmaster User's Guide* contains information and procedures for using the MEMORYmaster software to create embedded and distributed memories for use in ProASIC devices. This manual also includes information about all the possible configurations of RAMs and FIFOs that MEMORYmaster can generate. Furthermore, this manual includes brief examples of how to instantiate the generated memories into a design. Also information about timing for distributed memories as well as using multiple memories in a design is presented.

MEMORYmaster includes two independent tools: The Embedded MEMORYmaster generates memories using embedded memory blocks, and the Distributed MEMORYmaster generates distributed memories.

## Document Organization

The *MEMORYmaster User's Guide* is divided into the following chapters:

**Chapter 1- Embedded MEMORYmaster** lists all the possible configurations of embedded RAMs and FIFOs that MEMORYmaster can generate. It also describes using the memory generator and illustrates the different steps that are followed in two examples of a RAM and a FIFO.

**Chapter 2- Distributed MEMORYmaster** describes the architecture of distributed RAMs and FIFOs that are implemented in the logic tiles of ProASIC devices. The command line user interface is described and its use is illustrated on an example of a distributed RAM.

**Appendix A - Timing for Distributed Memories** presents the timing parameters for Level sensitive, Edge-triggered Register Files and FIFOs.

**Appendix B - Using Multiple Memories in a Design** provides an example of a design using one FIFO for receiving and another for sending data. Generation and integration of the FIFOs is also shown. Particular attention is given to their placement in embedded memory blocks of a device.

**Appendix C - Product Support** provides information about contacting Actel for customer and technical support.

# Document Assumptions

The information in this guide is based on the following assumptions:

1. You have installed the ASICmaster software.
2. You are familiar with the principles of digital design.
3. You are familiar with UNIX and Microsoft Windows NT operating systems

# ProASIC Series Manuals

This book is a part of the ProASIC series of manuals. All books in the series are listed below. Users can order these publications through an Actel sales representative.

*ASICmaster Installation and Licensing Guide* provides information and procedures for installing and licensing the ASICmaster software.

*ASICmaster User's Guide* provides information about the design flow for creating designs for ProASIC device. It includes information and procedures for placing and routing designs and also information on using timing constraints.

*MEMORYmaster User's Guide* provides information and procedures for generating embedded and distributed memories and instantiating then into a design.

*ProASIC Macro Library Guide* provides descriptions of ProASIC library elements for Actel's ProASIC device families. Symbols, truth tables, and timing parameters are included for all macros.

*ProASIC Interface Guide* provides information and procedures for designing Actel's ProASIC devices in Exemplar synthesis, Synopsys synthesis, Verilog simulation, and VHDL simulation environments.

# 1

# Embedded MEMORYmaster

This chapter describes how to use the Embedded MEMORYmaster to generate embedded memories for ProASIC devices, instantiate the memory into the design source code, simulate and synthesize the design, and import the netlist into ASICmaster. It includes a description of ProASIC dedicated memory blocks and all their possible configurations.

## Embedded Memory

ProASIC devices contain dedicated embedded memory blocks along with standard logic cells, called tiles as shown in the upper section of the ASIC master Viewer in Figure 1-1. below.

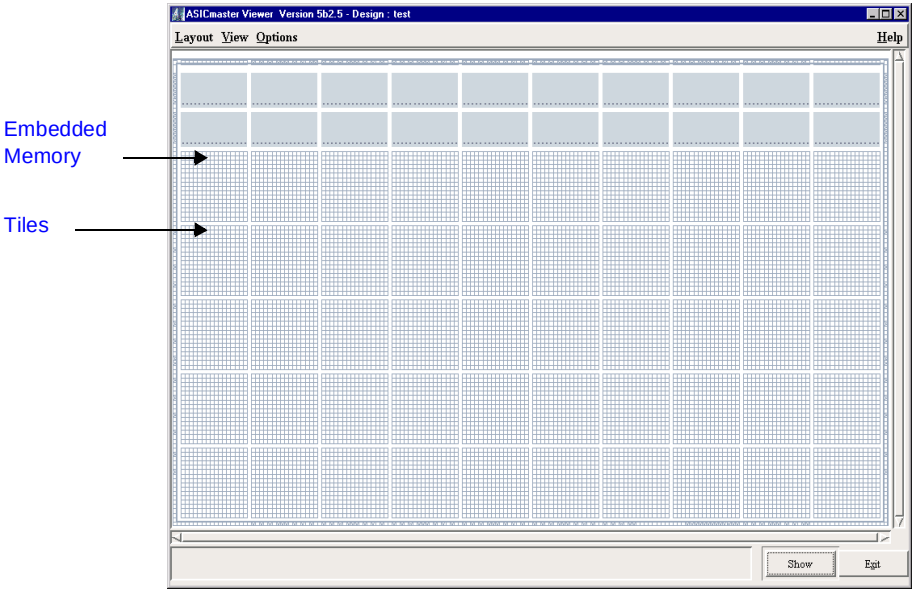


*Figure 1-1. Embedded Memory and Standard Logic Cells*

Each block can be configured to one of 24 functions as shown in Table 1-1 on page 2. Each memory block is 256 words deep and 9 bits wide,

for a total of 2304 bits of memory per basic memory block. Every memory block may be configured independently as two-port SRAM or as a FIFO. There are separate and independent read and write ports allowing simultaneous dual port access. The ports can be either synchronous or asynchronous, and each can be configured separately. This allows the option of using an asynchronous write and a synchronous read port. Synchronous output ports can be configured to either act like a transparent synchronous port or like a pipelined synchronous port. Additionally in all modes, a parity bit (9th bit) can be checked or generated within the memory. Parity check can be performed while writing and reading data without using additional logic. The result of these checks is shown in two independent signals WPE and RPE (Write Parity Error and Read Parity Error). Parity can also be generated while reading data.

## Embedded Memory Configurations

If allconfiguration possibilities are added up, 24 different memory modes can be configured.

*Table 1-1. Embedded Memory Block Configurations*

| Type | Write Access | Read Access | Parity | Library Cell Name |
|---|---|---|---|---|
| RAM | Asynchronous | Asynchronous Transparent | Checked | RAM256x9AA |
| RAM | Asynchronous | Asynchronous Transparent | Generated | RAM256x9AAP |
| RAM | Asynchronous | Synchronous Transparent | Checked | RAM256x9AST |
| RAM | Asynchronous | Synchronous Transparent | Generated | RAM256x9ASTP |
| RAM | Asynchronous | Synchronous Pipelined | Checked | RAM256x9ASR |
| RAM | Asynchronous | Synchronous Pipelined | Generated | RAM256x9ASRP |
| RAM | Synchronous | Asynchronous Transparent | Checked | RAM256x9SA |

*Table 1-1. Embedded Memory Block Configurations*

| | | | | |
|---|---|---|---|---|
| RAM | Synchronous | Asynchronous Transparent | Generated | RAM256x9SAP |
| RAM | Synchronous | Synchronous Transparent | Checked | RAM256x9SST |
| RAM | Synchronous | Synchronous Transparent | Generated | RAM256x9SSTP |
| RAM | Synchronous | Synchronous Pipelined | Checked | RAM256x9SSR |
| RAM | Synchronous | Synchronous Pipelined | Generated | RAM256x9SSRP |
| FIFO | Asynchronous | Asynchronous Transparent | Checked | FIFO256x9AA |
| FIFO | Asynchronous | Asynchronous Transparent | Generated | FIFO256x9AAP |
| FIFO | Asynchronous | Synchronous Transparent | Checked | FIFO256x9AST |
| FIFO | Asynchronous | Synchronous Transparent | Generated | FIFO256x9ASTP |
| FIFO | Asynchronous | Synchronous Pipelined | Checked | FIFO256x9ASR |
| FIFO | Asynchronous | Synchronous Pipelined | Generated | FIFO256x9ASRP |
| FIFO | Synchronous | Asynchronous Transparent | Checked | FIFO256x9SA |
| FIFO | Synchronous | Asynchronous Transparent | Generated | FIFO256x9SAP |
| FIFO | Synchronous | Synchronous Transparent | Checked | FIFO256x9SST |
| FIFO | Synchronous | Synchronous Transparent | Generated | FIFO256x9SSTP |
| FIFO | Synchronous | Synchronous Pipelined | Checked | FIFO256x9SSR |
| FIFO | Synchronous | Synchronous Pipelined | Generated | FIFO256x9SSRP |

The generation of some additional status signals besides the standard “EMPTY” and “FULL” signals is also built into the FIFOs. By providing a level signal, the circuit also generates signals that indicate whether the FIFO is filled less, equally filled, and filled higher than the specified level. For a description of how to connect the ports, and what function they have in each configuration see the *ProASIC Macro Library Guide*.

## Naming Conventions

The HDL Models for each of the 24 possible configurations, are included in the ProASIC simulation and synthesis library. The function and timing of each model is described in detail in the ProASIC *Macro Library Guide*. The modules are named according to the following convention:

```
<MEM-TYPE><256x9><WRITE-ACCESS><READ-ACCESS><PARITY>

  <MEM-TYPE>       := RAM or FIFO;
  <WRITE-ACCESS>   := A, S;
           A       := asynchronous;
           S       := synchronous;
  <READ-ACCESS>    := A, ST, SR;
           A       := asynchronous;
           ST      := synchronous transparent;
           SR      := synchronous registered;
  <PARITY>         := P or nothing;
           P       := parity will be generated;
           nothing := parity will be checked;
```

For example, the name of a FIFO with an asynchronous write and a synchronous transparent read mode with parity check is:

```
FIFO256x9AST.
```

Or a synchronous registered RAM with parity bit generation:

```
RAM256x9SSRP.
```

# MEMORYmaster Main Window

When you invoke MEMORYmaster, the main window, shown in Figure 1-2, is displayed. From this window, the format of embedded memory, the target ProASIC device and the configuration of the memory are selected.

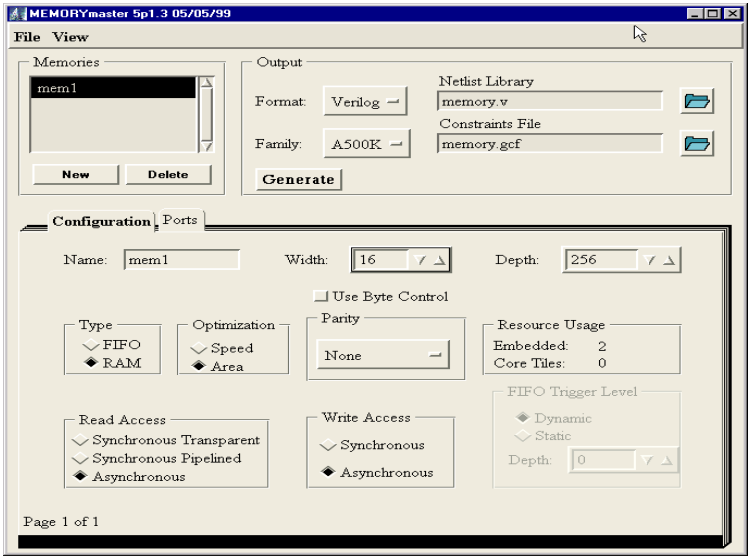


*Figure 1-2. MEMORYmaster Main Window*

The Configuration tab allows users to name the memory block currently being created and to select its parameters. Wide and deep RAMs and FIFOs can be created with a variety of options for access mode, parity, etc. One option available only for RAMs, is to write enable an individual byte that is wider than 8 bits. This is accomplished by enabling the appropriate write flag. One option only available for FIFOs is the ability to select the level signal as either static or dynamic. This level signal is used to set programmable flags. The value of this signal determines the occurrence of the EQTH and GEQTH signals.

The Resource Usage section is updated as changes are made to reflect the number of memory and logic tiles that the memory block requires in its implementation. The basic embedded 256x9 bit RAM or FIFO requires no logic core tiles. Logic core tiles are required for implementing embedded memories that are wider and/or deeper than 256x9 bit. The use of core resources is also dependent on the user's choice to generate a memory optimized for speed or area. Figure 1-2, shows a memory that was optimized for area. Optimizing for area required eight memory blocks and 29 core tiles. If it is optimized for speed the FIFO would be mapped to 8 memory blocks and 138 core tiles. Be aware that MEMORYmaster is not part specific. Users must ensure that the target device has sufficient memory resources to accommodate the memory blocks. For example, the A500K130 has 20 basic 256x9 bit memory resources available. If a 512x32 bit FIFO is defined using MEMORYmaster, then every instance of that memory will use eight of the available memory resources.

The Ports tab, shown in Figure 1-3, allows the specification of names for ports used in the netlist. Default names are automatically provided. Only those signal fields relevant to the selected memory type will be active.

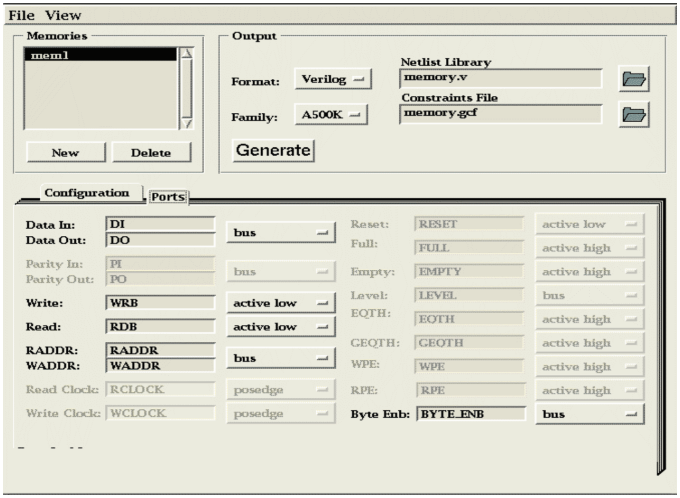


*Figure 1-3.* MEMORYmaster Window Ports Tab

The bus switch next to the Data In and Data Out fields allows users to specify whether to treat these signals as bus or signal bit ports. Users can also select whether control signals are active high or active low. The byte enable button allows users to select any arbitrary byte in the RAM (for word width > 8) as active during a write cycle. If selected, the RAMs will implement at most 8 bits of the word.

# Generating Embedded Memories

MEMORYmaster generates memories in Verilog, VHDL or EDIF format. One memory block or multiple blocks can be generated, and the blocks may reside in one or more netlist files.

A constraints file, containing placement directives for each memory, is also generated. The constraint data for each block may reside in one or more constraint files.

Use the following procedure to generate an embedded memory with Embedded MEMORYmaster.

1. **Invoke Embedded Memorymaster.**

   **PC**

   Select ASICmaster5p1 from the ASICmaster menu under programs in the Start menu

   **UNIX**

   Type the following command at the prompt:

   ```
   asicmaster
   ```

2. **From the ASICmaster launch Window select the Toolbar click the Tools button.**


*Figure 1-4. ASICmaster Toolbar*

3. **Click the Build Embedded Memories button from the ASICmaster Tools dialog box shown in Figure 1-5 on page 8.**


*Figure 1-5. ASICmaster Tools Dialog Box*

Note: On the PC you must first set a working directory for your design and click the Select button as shown in Figure 1-6. On Unix simply click Build Embedded Memories and the Embedded MEMORYmaster Main window is displayed.

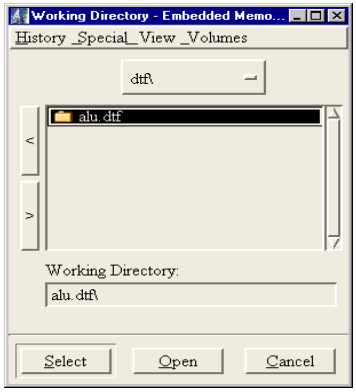


*Figure 1-6. Working Directory File Browser*

4. **Once the Embedded MEMORYmaster Main window is displayed as shown in Figure 1-2 on page 5, click New.** A new memory configuration is created named mem1.
5. **In the Output field, specify the format of the netlist to be created.**
6. **On the Configuration tab, shown in Figure 1-2, name the memory block currently being created in the Name field, and select its parameters from the choices given.**
7. **On Ports tab, shown in Figure 1-3, specify the names for ports used in the netlist.** Default names are automatically provided. Only those signal fields relevant to the selected memory type will be active
8. **When you are satisfied with the current settings, click the Generate button.** MEMORYmaster will create the netlist and constraint files. To specify additional memory blocks, click the New button. If the same netlist and constraint file names are used as in a previous memory definition, the data for the new block will be appended to those files.

   Note: If the named files already exist, their contents will be overwritten with the current memory data.

## Integrating Memories into a Design

This section provides one example of how to instantiate a Verilog, and a VHDL netlist, link it to a design.

Once MEMORYmaster has generated the needed memories you must instantiate the netlist into your design before simulating and synthesizing. The tool either generates a memory.v, memory.vhd or memory.edif netlist file and a memory.gcf constraints file.

## Verilog Example RAM 512x32

The following is a Verilog netlist generated from MEMORYmaster for a 512x32 bit RAM:

```
`timescale 1ns/10ps
// type = RAM
// width = 32
// depth = 512
// output type = transparent
// parity control = ignore
// parity type = odd

module ram512x32(DO, CLOCK, DI, WRB, RDB, WADDR, RADDR);
  output [31:0] DO;
  input CLOCK;
  input [31:0] DI;
  input WRB;
  input RDB;
  input [8:0] WADDR;
  input [8:0] RADDR;
  ...
```

The following is an example of how to instantiate a RAM512X32 module into a design:

```
ram512x32 MY_RAM_INST(.DO(data_out),.CLOCK(clk), .DI(data_in),
  .WRB(wrb), .RDB(rdb),.WADDR(write_add), .RADDR(read_add));
```

After adding the instantiation of the memory into the Verilog source code, the next step is to synthesize the design. Before synthesizing the design, make sure that the “dont_touch” attribute is set on all memories generated by MEMORYmaster. Refer to the documentation included with your synthesis tool for further information.

## VHDL RAM Example

The following is a VHDL example of the previous generated memory:

```
entity ram512x32 is
port( DO      :out  std_logic_vector (31 downto 0);
      CLOCK   : in std_logic;
      DI      : in std_logic_vector (31 downto 0);
      WRB     : in std_logic;
      RDB     : in std_logic;
      WADDR   : in std_logic_vector (8 downto 0);
      RADDR   : in std_logic_vector (8 downto 0)
      );
end ram512x32;
```

The entity describes the interface of the module that must be instantiated into the VHDL design source code. Besides the actual connection of the interface, VHDL requires an additional declaration of the sub-module in the architecture. The following is an example of an architecture including the declaration of the memory as a component:

```
architecture MY_DESIGN_STRUCTURE of MY_DESIGN is
component RAM512x32
port( DO      : outstd_logic_vector (31 downto 0);
      CLOCK   : in std_logic;
      DI      : in std_logic_vector (31 downto 0);
      WRB     : in std_logic;
      RDB     : in std_logic;
      WADDR   : in std_logic_vector (8 downto 0);
      RADDR   : in std_logic_vector (8 downto 0)
      );
end component;


begin
...

MY_RAM : RAM512x32 Port Map(DO  =>data_out,
                            CLOCK   =>clk,
                            DI      =>data_in,
                            WRB     =>wrb,
                            RDB     => rdb,
                            WADDR   => write_add,
                            RADDR   => read_add
                            );
...
end MY_DESIGN_STRUCTURE;
```

## Importing the Netlist

After synthesis a design is translated into either a Verilog, VHDL or EDIF netlist. The netlist includes all logic blocks as well as the memories. To import the netlist file(s) into ASICmaster refer to the *ASICmaster User's Guide*. Placement information generated by MEMORYmaster must be imported into ASICmaster as constraints. This enables ASICmaster to place the memories automatically.

*Table 1-2. Possible RAM Locations for the A500K Family*

| Part | possible RAM locations | formula |
|---|---|---|
| A500K050 | (1,57), (17, 57), ..., (81, 57) | x = 16*n+1; n= {0,1,2,3,4,5}; y = 57; |
| A500K130 | (1,81), (17, 81), ..., (145,81) (1,89), (17, 89), ..., (145,89) | x = 16*n+1; n= {0,1,2,3,4,5,6,7,8,9} y = {81, 89} |
| A500K180 | (1,97), (17,97), ..., (177, 97) (1,105), (17,105), ..., (177, 105) | x = 16*n+1; n= {0,1,2,3,4,5,6,7,8,9,10,11} y = {97, 105} |
| A500K270 | (1,121), (17,121), ..., (209,121) (1,129), (17,129), ..., (209,129) | x = 16*n+1; n= {0,1,2,3,4,5,6,7,8,9,10,11,12,13} y = {121, 129} |

The placement constraints file contains statements such as the following:

```
memory <mem_name> (
(M3 M1)
(M2 MO)
macros_to_place (<macro1> <macro2> )
)
```

MEMORYmaster may generate additional macros in the placement contraints file to place the additional logic needed for deeper and or wider RAMs/FIFOs.

Note: If you use these memory modules in a synthesis flow, ensure that you set "dont_touch" attributes on the modules generated by MEMORYmaster. Otherwise, the names of these modules may be changed and ASICmaster cannot find the memory modules to be placed in the netlist.

## Manual Memory Placement

For manual placement, a constraints file must be created. The following is an example of a manually created placement file for a 500k130 device.

```
set_location (1,81) <hier_instance_name>/M0;
set_location (1,89))<hier_instance_name>/M1;
set_location (33,89)<hier_instance_name>/M2;
set_location (33,81) <hier_instance_name>/M3;
```

The (x,y) coordinates changes for each device. If wrong coordinates are entered, ASICmaster reports about wrong coordinates and displays a list of valid coordinates for the selected device.

# Embedded Memory Examples

This section provides examples of how to generate memory and integrate it into your design. These examples also briefly explain simulation, synthesis and place and route for each example:

## Example 1- Two Port Memory

This example demonstrates how to generate a two port RAM that is 1024 words deep by 8 bits wide with synchronous write and asynchronous read and an even parity generation.

### Generate the Memory

1. **Invoke ASICmaster**.
2. **Click the Tools button**. The ASICmaster tools dialog box is displayed.
3. **Click Build Embedded Memories.** The MEMORYmaster main window is displayed (see Figure 1-5 on page 8).

   Note: On Windows NT, you need to selct a corking directory first.
4. **Click the New button.** All fields are activated.
5. **Specify the name of memory in the name box.** Use “mem1024x8” for this example.
6. **Specify RAM as the Type.**
7. **Specify Synchronous as the Write Access.**
8. **Specify Asynchronous as the Read Access.**
9. **Type 1024 as the Depth and 8 as the Width.** (After selecting the other required fields, the MEMORYmaster main window will appear as in Figure 1-7 on page 16).
10. **Specify Generate Even as the Parity.**
11. **If you would like to assign special names to memory ports, click the Port Tab and enter the desired names.**
12. **Click the Generate button.**

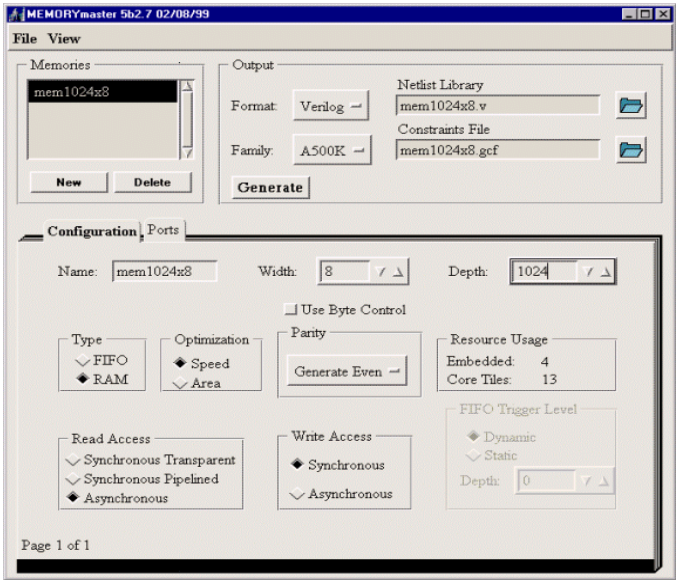


*Figure 1-7. MEMORYmaster Main Window*

### Integrate the Memory into a Design

The following code shows how to instantiate the generated memory in Verilog:

```
module Top_level(in,out clk,reset……..);
input in, clk reset;
output out;
....

mem1024x8 MEM1(.DI(wr_data), .DO(rd_data)……);
// This is memory instantiation

always@(posedge clk or reset or in)
    begin
....
    end;
endmodule
```

### Simulate and Synthesize

After instantiating a memory into the design, you can simulate and synthesize it. Memory models are included into the simulation and synthesis libraries. Refer to the documentation included with your simulation and synthesis tools for additional information. Make sure that the "dont_touch" attribute is set on all memories generated by MEMORYmaster during synthesis. The following is an example Verilog simulation command:

```
verilog test_sim.v  top_level.v  mem1024x8.v  -v  $AMHOME/etc/
deskits/verilog/lib/A500K.v
```

You must link to the simulation libraries that are located under: `$AMHOME/etc/deskits/verilog/lib/A500K.v`.

### Place and Route

After synthesis, the netlist contains the embedded memories. For automatic placement, use the constraints file (mem1024x8.gcf) generated by MEMORYmaster. If automatic placement is used, memories are placed in a line and can be placed and routed in ASICMASTER. For manual placement, create a text file such as the following, and read this file into ASICmaster:

```
set_location  (1,81) MEM1/M0;
set_location  (17,81) MEM1/M1;
set_location  (33,81) MEM1/M2;
set_location  (49,81) MEM1/M3;
```

Memory blocks can in any legal position. See Table 1-2 on page 13 for legal position of memory blocks.

## Example 2-512x16 FIFO

This example demonstrates how to generate an asynchronous FIFO that is 512 deep by 16 wide with FULL, EMPTY, and HALF FULL flags and how to integrate it into a design.

### Generate the FIFO

1. **Invoke ASICmaster**.
2. **Click the Tools button**. The ASICmaster tools dialog box is displayed.
3. **Click Build Embedded Memories. The MEMORYmaster main window is displayed**
4. **Specify all required fields for the design.** Specify static as the FIFO Trigger Level and half depth for the programmable flag **.** Enter width and depth. Select FIFO under Type field.
5. **Click generate.**

### Instantiate the FIFO into the design

The following code shows how to instantiate the memory in Verilog:

```
module top_level (in,out clk,reset……..);
input in....;
........

FIFO512x16 rec_fifo(.DI(wr_data), .DO(rd_data)……);
// This is fifo instantiation

always@(posedge clk or……..

........

endmodule
```

### Simulate and Synthesize

After instantiating a FIFO into the design, you can simulate and synthesize it. Memory models are included into the simulation and synthesis libraries. Refer to the documentation included with your simulation and synthesis tools for additional information. Make sure that the “dont_touch” attribute is set on all memories generated by MEMORYmaster during synthesis.

The following is an example Verilog simulation command:

```
verilog test_sim.v  top_level.v FIFO512x16.v  -v  $AMHOME/etc/
deskits/verilog/lib/A500K.v
```

You must link to the simulation libraries that are located under: `$AMHOME/etc/deskits/verilog/lib/A500K.v`.

### Place and Route

After synthesis, the netlist contains the embedded memories. For automatic placement, use the constraints file (mem512x16.gcf) generated by MEMORYmaster. If automatic placement is used, memories are placed in a line and can be placed and routed in ASICMmaster. For manual placement, create a text file such as the following and read this file into ASICmaster:

```
set_location  (1,81) MEM1/M0;
set_location  (17,81) recfifo/M1;
set_location  (33,81) recfifo/M2;
set_location  (49,81) recfifo/M3;
```

Memory blocks can be in any legal position. See Table 1-2 on page 13 for legal position of memory blocks.

## Multiple Memory Generation and Integration

If a design includes several memories with different sizes and access modes, Actel recommends generating them all in one session of MEMORYmaster. The embedded memories are automatically generated and are accompanied by placement directives. ASICmaster uses these directives to place the memories efficiently. Appendix B, “Using Multiple Memories in a Design” on page 35, shows two examples of designs using multiple memories and illustrates their placement in a ProASIC device.

# 2
# Distributed MEMORYmaster

This chapter describes how to use the Distributed MEMORYmaster to create distributed memories for ProASIC device.

## Distributed Memory Architecture

Distributed memory can be generated as a two port asynchronous register file or as an asynchronous FIFO. Distributed memories are placed in the logic tiles of the device. Therefore, these memory files are netlists consisting of logic tiles and do not refer to the normal ProASIC memory libraries.

### The Register File

The register file has independent read and write ports. The read port is asynchronous so the read data is not clocked and is available a short time after the read address changes. The write operation can be either level sensitive or edge-sensitive. The schematic of a 2x2 memory is shown in Figure 2-1 on page 22. The schematic is marked to show the words (vertical slices), the bits (horizontal slices) and the decoders (one per word). The register file memory requires 1 column per word and 2 rows per bit plus from 1 to 3 rows for the necessary decoders.

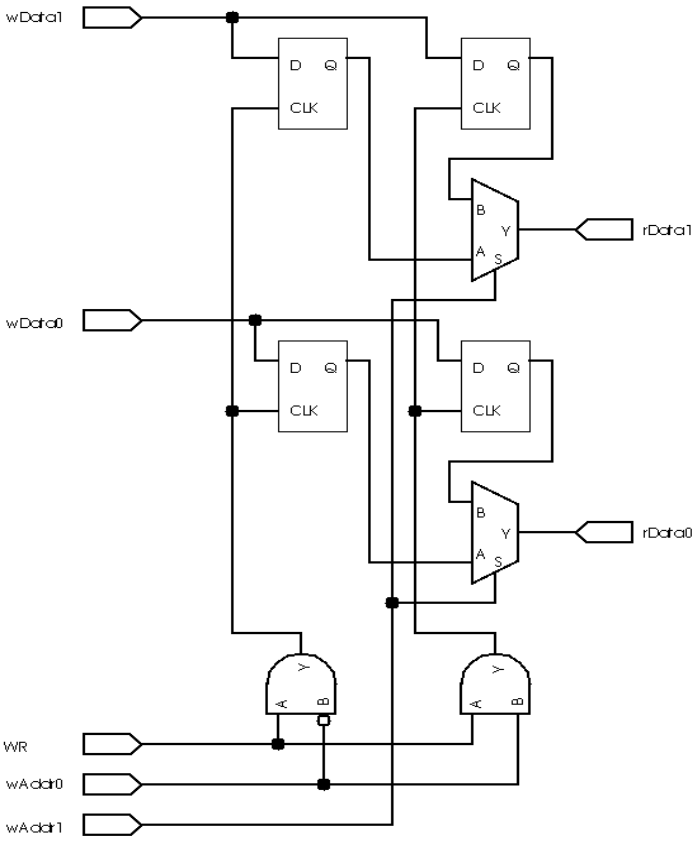


*Figure 2-1. Schematic of a 2x2 Register File*

## Distributed FIFO

Distributed FIFO memory has independent read and write ports. However, it has no address ports. Instead, the FIFO keeps track of the addresses internally. The FIFO is organized with words in columns and data bits in rows. The top row consists of the write addressing circuitry

and the “full” detection circuitry. The bottom row consists of the read addressing circuitry and the “empty” detection circuitry. The FIFO memory requires 2 columns per word plus an overhead for decoders and flag generation that is a minimum of 3 columns. The FIFO also requires 1 row per bit plus an overhead of 2 rows. Figure 2-2 shows the schematic of a 2x2 FIFO.

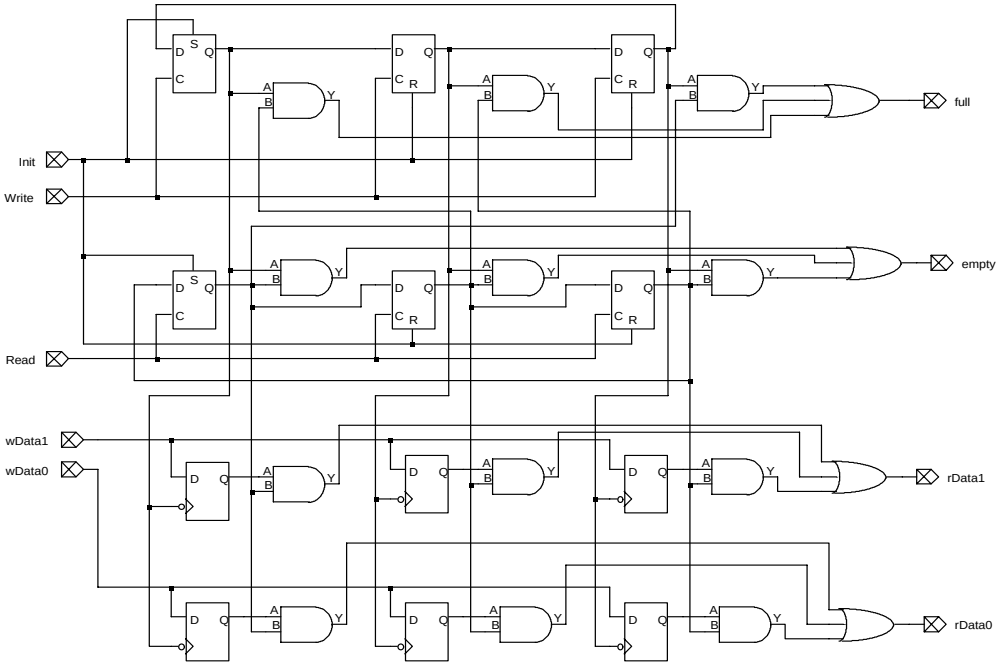


*Figure 2-2. Schematic of a 2x2 FIFO*

# Determining Tile Usage

ProASIC parts tend to have more tiles horizontally. The choice of orientation affects the allowable size of the memory. A horizontal memory allows the maximum possible number of words, whereas a vertical memory allows the maximum number of bits-per-word. MEMORYmaster can create register files of up to 64 words on any possible ProASIC device. Distributed memories are created using logic tiles and they are generally slower and larger compared to embedded RAM. Actel recommends that larger memories be implemented with the embedded memories. The maximum distributed FIFO sizes in any ProASIC device is 80 words. The maximum FIFO sizes are shown in Table 2-1.

*Table 2-1. Maximum FIFO Dimensions*

| Device | Vertical | | Horizontal | |
|---|---|---|---|---|
| | **Words** | **Width** | **Words** | **Width** |
| A500k050 | 64 (23) | 46 (94) | 64 (36) | 30 (62) |
| A500k130 | 64 (29) | 46 (158) | 64 (62) | 30 (78) |
| A500k180 | 64 (36) | 46 (192) | 64 (75) | 30 (95) |
| A500k270 | 64 | 46 (222) | 64 (80) | 30 (118) |

The orientation of the register file affects how it is placed. Horizontal register files are placed with words in columns and bits in rows as shown in Figure 2-3.

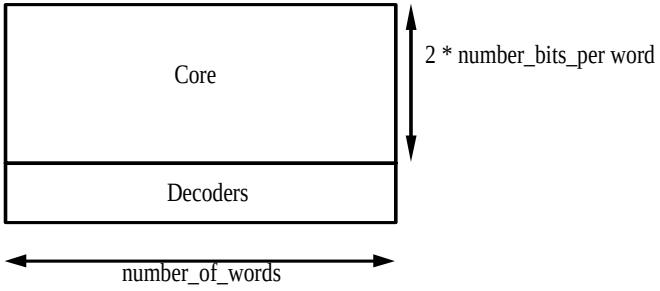


*Figure 2-3. Horizontal Memory*

The memory size is then the number-of-words wide by 2*number of bits per word+decoder_size high. Alternatively, if the memory orientation is vertical then the words are arranged in rows and the bits in columns. The memory size is then 2*number of bits per word+decoder_size wide by number-of-words high as shown in Figure 2-4 on page 26. The decoder sizes are given in the following table:

*Table 2-2. Decoders Sizes*

| Number of Words | Decoder Size |
|---|---|
| 2 ~ 4 | 1 |
| 5 ~ 8 | 2 |
| 9~64 | 3 |

The following is a an example logic usage calculation for a vertical RAM16x32:

Width: 2*number-of-bits-per-word+decoder_size = 2 * 32 + 3 = 67 tiles

Height: number-of-words = 16 = 16 tiles

Example RAM16x32 horizontal:

Width: number-of-words = 16 = 16 tiles

Height: 2*number-of-bits-per-word+decoder_size = 2 * 32 + 3 = 67 tiles

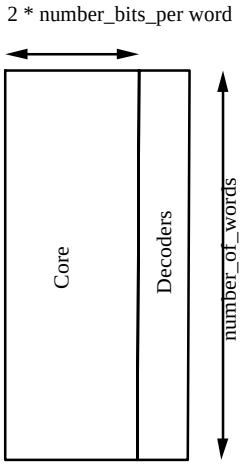


*Figure 2-4. Vertical Memory*

Finally, the tool displays the legal coordinates to place the memory if the macro is not rotated or flipped. In this case a placement of the macro between the coordinates (1,1) and (31,49) would cause no problem.

### Distributed Memory Placement

To achieve the best timing and efficient placement, use the placement constraints file generated by the Distributed MEMORYmaster. For more information on constraint statements, refer to the *ASICmaster User's Guide*. To utilize this file, use the set_location constraint statement for macros.

```
set_location (x,y) <mem_hier_name> <macro_name>;
```

### Distributed Memory Timing

Memory timing values are dependent on the memory size and the routing to and from the memory. Since the memories are implemented as ProASIC primitives, users can determine the timing characteristics of the circuit by performing a backannotated timing analysis. In fact, to the timing analyzer, the distributed memory looks like any other part of the circuit and requires no special treatment. The first sections in Appendix A explains the critical timing paths in each memory, and why these paths are critical.

## Distributed Memory Generation and Instantiation

Consider the following hierarchical design, which instantiates a 16x32 memory as shown in Table 2-5.

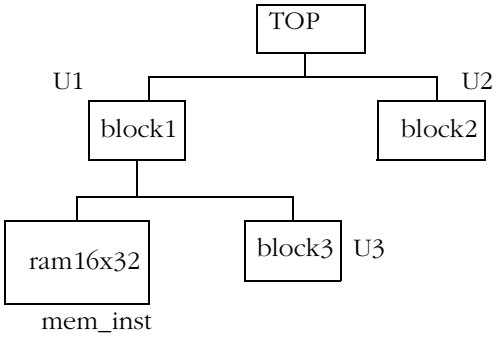


*Figure 2-5. Design Example*

Figure 2-6 shows how to generate a RAM of 16x32 using the Distributed MEMORYmaster tool. Use the generated file to instantiate the RAM into your design.

```
Welcome to ASICmaster's Distributed MEMORYmaster Version
V5p1
(C) 1996-1999 Gatefield Corporation. All rights reserved.
Note:   See the MEMORYmaster User Guide for detailed
        information about this program and how to use it.
Please enter output format:
        1 for verilog
        2 for vhdl ?1
Type of module:
        1 for 2-port asynchronous register file
        2 for fifo
        0 to quit? 1
Type of device:
        1 for A500K050
        2 for A500K130
        3 for A500K180
        4 for A500K270
        0 to quit?1
Orientation of memory:
        1 for horizontal
        2 for vertical
        0 to quit? 2
Cell name? ram16x32
Number of words (2...64)? 16
Number of data bits per word (2...46)? 32
Memory type:
            'e' for edge triggered,
            'l' for level-sensitve? e
Generating verilog file "ram16x32.v" in "/home1/users"
Generating gcf file "ram16x32.gcf" in "/home1/users"
Legal X: (1...31)
Legal Y: (1...49)
Done.
Type <cr> to close the window.
```

*Figure 2-6. Distributed MEMORYmaster User Interface*

### Simulation and Synthesis

After instantiating the memory into the design, you can follow the same design flow as for embedded memories. This memory is made of logic cells and can be simulated by linking the simulation libraries. The following is an example Verilog simulation command:

```
verilog test_sim.v top.v mem16x32.v -v $AMHOME/etc/deskits/
verilog/lib/A500K.v
```

The design can be synthesized, but the “dont_touch” attribute must be set for memory blocks during synthesis.

### Place and Route:

During place and route users must use placement constraints generated by MEMORYmaster. The ram16x32.gcf file must be imported as a constraint file into ASICmaster and the following statement must be included in a text file.

```
set_location (10,10) U1/mem_inst mem16x32;
```

Read this text file also into ASICmaster. Now ASICmaster treats the memory as a macro and places memory in a rectangle with the bottom-left corner on tile coordinate (10,10). Memory can be moved on the die by changing this coordinate.

Note: Distributed memory contains very high fanout nets so if you do not use the above placement constraints, memory timing will be sub-optimal or the design may not route.

# 3

# Timing for Distributed Memories

The following chapter decribes the timing parameters for the level sensitive register file, and edge-triggered register file. Also it include is information about Edge-triggered FIFOs

## Level-sensitive Register File

The level-sensitive register file has three main timing parameters:

Tacc - time from stable read-address to output data valid

Tsetup_data - time from stable write-data to falling clock edge

Tsetup_addr - time from stable write-address to rising clock edge

Figure 3-1 shows these relationships:

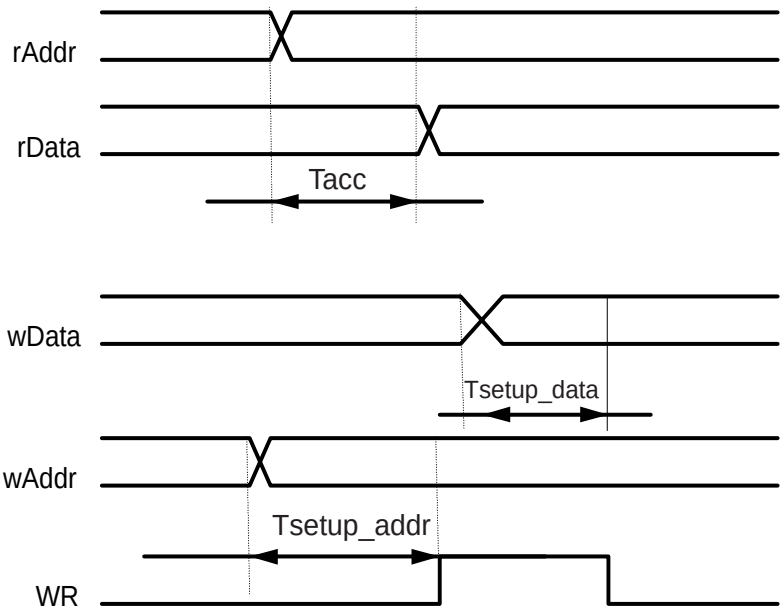


*Figure 3-1. Level-sensitive Mode Timing Diagram*

Failure to meet these timing values will have the following results:

Tacc - read data might be from previous address

Tsetup_data - data may not be written into the memory

Tsetup_addr - data may be written into some other address as well as the intended address

## Edge-triggered Register File

The edge-triggered register file has three main timing parameters:

Tacc - time from stable read-address to output data valid

Tsetup_data - time from stable write-data to rising clock edge

Tsetup_addr - time from stable write-address to rising clock edge

Figure 3-2 shows the relationships of the signals:

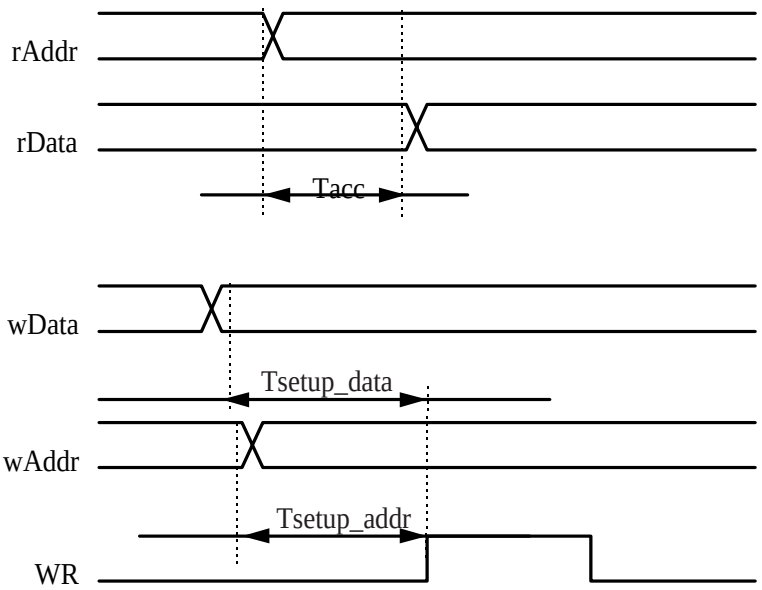


*Figure 3-2. Edge-triggered Mode Timing Diagram*

Failure to meet these timing values will have the following results:

Tacc - read data might be from previous address

Tsetup_data - data may not be written into the memory

Tsetup_addr - data may be written into some other address

The main advantage of the edge-triggered memory is that the write timing is sensitive only to the rising edge of the WR, not both the rising and falling edges.

## Edge-triggered FIFO

The edge-triggered FIFO captures data on the rising edge of the WR signal, and the read pointers advance on the rising edge of the RD signal. Before using the FIFO, it must be initialized by pulsing the INIT signal high. Immediately after initialization, the “empty” signal is true and the “full” signal false. Data applied on the wDataX lines are captured when the WR signal transitions from 0 to 1. Simultaneously, the “empty' signal will become false to indicate that there is valid data on rDataX. Further transitions from 0 to 1 on WR captures more data into the FIFO until such time as “full” becomes true. At this point, the FIFO is full, and no more data should be entered into it.

Afte the FIFO is initialized, the output data remains invalid until the first read operation is performed. With every rising edge of the read pulse, the FIFO generates the next word written into it on the output data bus until all the words written into it are read out. At this point the empty signal goes high. Further read operations produces no change to the data output as it remains fixed at the last word written into the FIFO.

Figure 3-3 shows an example of an initialized FIFO, ten words are written, and read.

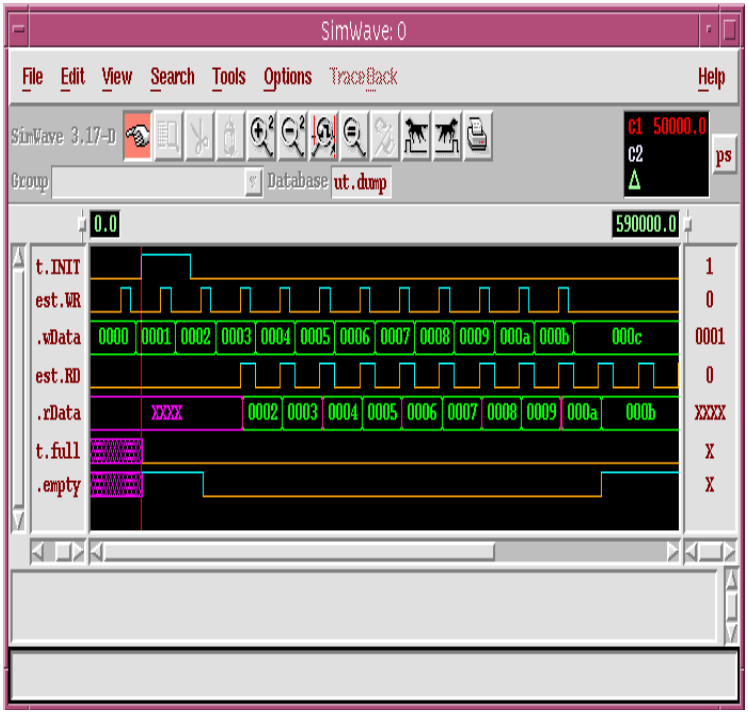


*Figure 3-3. Edge-triggered FIFO*

## Level Sensitive FIFO

The level sensitive FIFO has the same timing as the edge-triggered FIFO. The only difference is that the data input is latched at the falling edge of the write pulse.

# 4

# Using Multiple Memories in a Design

This chapter describes how to use multiple memories in a design. ProASIC devices contain dedicated memory blocks that can be configured as RAM or FIFOs. Leaf memory block have certain control signals and multiple memory blocks can be combined together to create deep and wide memories. MEMORYmaster does this by combining multiple memory blocks as required. MEMORYmaster generates a netlist and placement constraints. Netlist instantiates memory leaf cells and the placement constraints file contains placement information for each memory leaf cell. During place and route, this information can be used to place memory blocks automatically. These placement constraints can also be used if the design is instantiating multiple memories. Consider the following design shown in Figure 4-1.

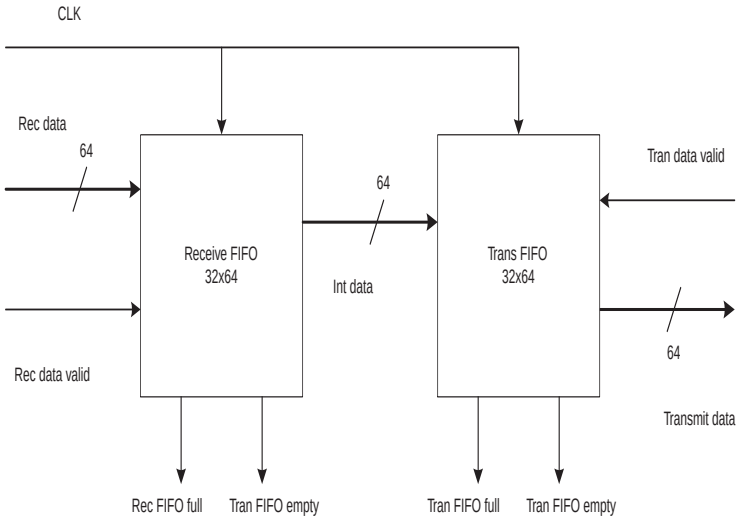


*Figure 4-1. Sample Design*

In this design, there is a receive FIFO and transmit FIFO. Read and Write ports are synchronous. Each FIFO is 32 words deep and 64 bits wide. Also, both FIFOs are identical. Only one FIFO needs to be created with MEMORYmaster, and it must be instantiated twice into the design.

The MEMORYmaster window that illustrates how to create this FIFO is shown in Figure 4-2.

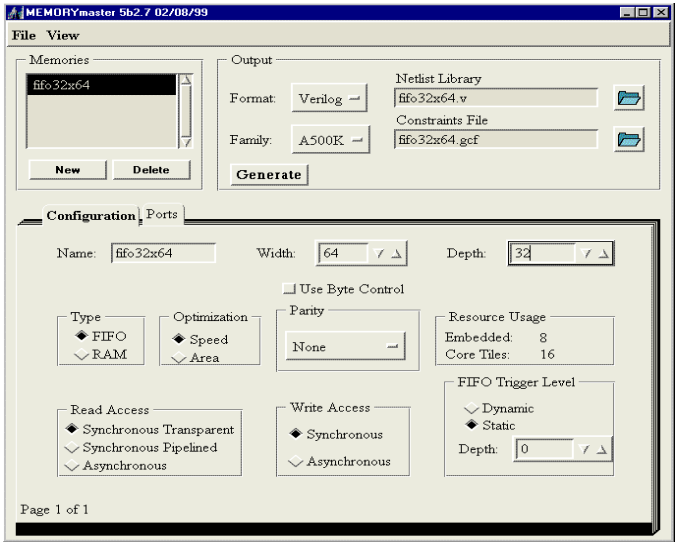


*Figure 4-2. MEMORYmaster Window Showing a Sample FIFO*

Once the FIFO is generated with MEMORYmaster, it must be instantiated into the design. The following is an example of the RTL after instantiation:

```
module top(tran_data, rec_data, rec_data_valid,
tran_data_valid, clk,
    reset, rec_fifo_full, rec_fifo_empty,tran_fifo_full,
tran_fifo_empty);  // this is top level module
input rec_data_valid, clk, reset, tran_data_valid;
output[63:0] tran_data;
output rec_fifo_full, rec_fifo_empty,tran_fifo_full,
tran_fifo_empty;
input[63:0] rec_data;
wire[63:0] data_int;
/* Reciever FIFO instantiation */
sync_fifo rec_FI(.data_in(rec_data),.data_out(data_int),
                 .wr(rec_data_valid), .rd(1'b0),
                 .empty(rec_fifo_empty), .full(rec_fifo_full),
```

```
                .reset(reset),.clk(clk));
/* transmit FIFO instantiation */
sync_fifo tran_F1(.data_in(data_int), .data_out
(tran_fifo_full)
              .wr(1'b0),.rd(tran_data_valid),
              .empty(tran_fifo_empty),.full(tran_fifo_full),
              .reset(reset), .clk(clk));
/* other RTL of the design and other blocks */

endmodule


module sync_fifo (data_in, data_out, wr,
rd,empty,full,reset,clk);
input[63:0] data_in;
output[63:0] data_out;
input wr, rd,clk,reset;
output empty,full;
/* Instantiation of FIFO generated from MEMORYmaster */
fifo32x64 F1(.DO(data_out), .RCLOCK(clk), .WCLOCK(clk),
            .DI(data_in), .WRB(wr), .RDB(rd), .RESET(reset),
            .FULL(full), .EMPTY(empty), .EQTH(), .GEQTH());


endmodule
```

## Simulate and Synthesize

At this level, you can simulate and synthesize the design.The following is and example of a verilog simulation command:

```
verilog test_sim.v  top.v  fifo32x64.v –v  $AMHOME/etc/des-
kits/verilog/lib/A500K.v
```

The following is a typical synthesis script for Synopsys Design Compiler:

```
read –format verilog fifo32x64.v
set_dont_touch find(design, “fifo32x64.v”)  /* memories must
be dont_touch during synthesis */
read –format verilog top.v
create_clock –period 20 clk  /* add timing constraints */
set_wire_load A500K
set_operating_conditions WORST
compile
set_port_is_pad “*” /* use set_pad_type to to use a particular
type of pad */
insert_pads
```

```
write -format verilog -hierarchy -output top_str.v  /* write
out netlist with hierarchy */
quit
```

The netlist top_str.v contains both FIFO instantiations and can be used for simulation for post synthesis gate level simulation.

After synthesis, you can place and route the design. If there are multiple instances of the same memory, ASICmaster automatically determines that the same constraints file applies to all memories. So, place and route should be run with top_str.v as a netlist and fifo32x64.gcf as a constraints file. In this example, each FIFO uses 8 memory blocks. ASICmaster attempts to place each FIFO in a line. The resulting placement on an A500K130 device, which has 20 memory slots is shown in Figure 4-3.

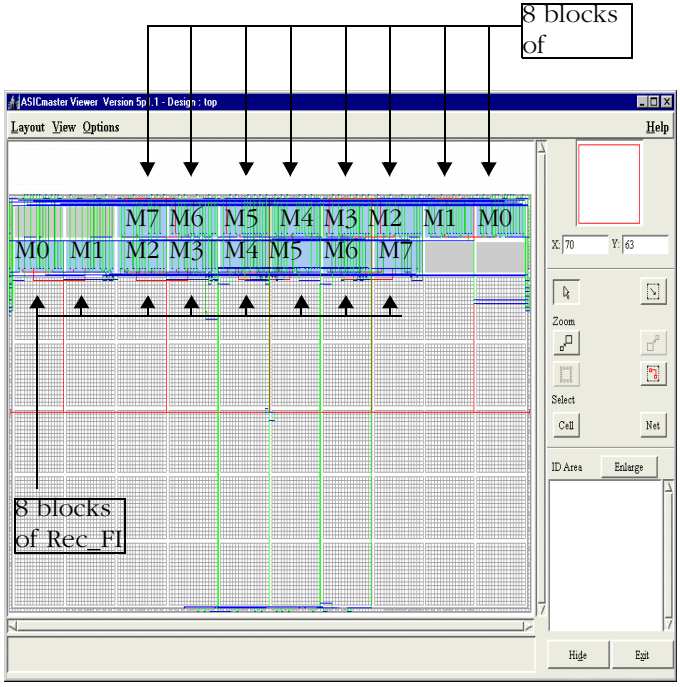


*Figure 4-3. Sample Memory Placement*

During placement ASICmaster attempts to keep one memory entity in one group. In this example, it placed the Rec_FI/F1/M0 in the first memory slot in the left side of the lower row, and rec_FI/F1/M1 in next slot and so on. Only ten slots are available in one row, therefore, trans FIFO placement started from the upper row. If each memory block had used four blocks, both memory blocks would be placed one after another in the lower row.

## Manual Placement of Multiple Memories

A memory placement file must be created to manually place memories. For example, to place the receive FIFO from the previous example on the left side using both rows and the transmit FIFO on right side in both rows, the following placement file would be used:

```
set_location (1,81) rec_FI/F1/M0;
set_location (1,89) rec_FI/F1/M1;
set_location (17,89) rec_FI/F1/M2;
set_location (17,81) rec_FI/F1/M3;
set_location (33,81) rec_FI/F1/M4;
set_location (33,89) rec_FI/F1/M5;
set_location (49,89) rec_FI/F1/M6;
set_location (49,81) rec_FI/F1/M7;


set_location (145,81) tran_FI/F1/M0;
set_location (145,89) tran_FI/F1/M1;
set_location (129,89) tran_FI/F1/M2;
set_location (129,81) tran_FI/F1/M3;
set_location (113,81) tran_FI/F1/M4;
set_location (113,89) tran_FI/F1/M5;
set_location (97,89) tran_FI/F1/M6;
set_location (97,81) tran_FI/F1/M7;
```

This constraints file should be read into ASICmaster and will result in the placement as shown in Figure 4-4 on page 40.

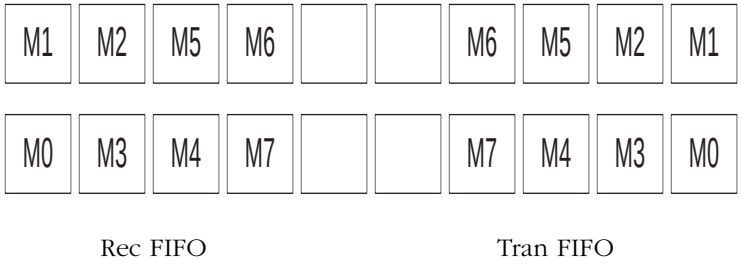


*Figure 4-4. Sample FIFO Placement*

## Multiple Instances of Various Memories

MEMORYmaster will generate constraint files for each memory and all of these files should be read into ASICmaster during place and route. ASICmaster determines the placement for each memory and will keep each memory entity together. For automatic placement of memories, it is recommended that all constraint files from MEMORYmaster be read into ASICmaster. To change default placement, you can discard constraints from MEMORYmaster and create your own for memory placement.

If very deep or very wide memories are created, MEMORYmaster combines together multiple blocks and uses glue logic to combine them. Two lists quantifying glue logic are shown in Table 4-1 and Table 4-2 on page 41 .

These tables cover extreme cases of depth or width for RAM and FIFO for the A500K130 device, which has 20 memory blocks and 12800 logic tiles.

*Table 4-1. RAM*

| RAM | **Parity** | **Memory Blocks Used** | **Logic Tile Used** | **Comment** |
|---|---|---|---|---|
| Depth 5120 Width 8 | Check Even | 20 | 257 | All 20 blocks used in depth |
| Depth 256 Width 160 | Check Even | 20 | 2 | All 20 blocks used in width |

*Table 4-2. FIFO*

| **FIFO** | **Parity** | **Memory Blocks Used** | **Logic Tile Used** | **Comment** |
|---|---|---|---|---|
| Depth 5120 Width 8 | Check Even | 20 | 590 | All 20 blocks used in depth |
| Depth 256 Width 160 | Check Even | 20 | 60 | All 20 blocks used in width |

For FIFOs, MEMORYmaster creates placement directives for glue logic. If placement information from MEMORYmaster is used, glue logic placement is more efficient.

# Programmable Flag in FIFOs

ProASIC devices provide a programmable flag for FIFOs. The threshold for this flag can be set in MEMORYmaster in the main menu, shown in Figure 1-7 on page 16. It is on the right bottom corner in the 'FIFO Trigger Level box. You can specify whether the flag is static or dynamic. If it dynamic is selected, MEMORYmaster will create memory with a LEVEL bus on the memory interface. Consequently, you can apply any value to this bus to change its threshold dynamically. If the threshold is not changing, you can select the static option and specify the threshold value. In this case, Memorymaster will hardwire threshold to the specified value. A detailed timing of these flags can be found in the A500k *ProASIC Datasheet*.

# A

# Product Support

Actel backs its products with various support services including Customer Service, a Customer Applications Center, a Web and FTP site, electronic mail, and worldwide sales offices. This appendix contains information about using these services and contacting Actel for service and support.

## Actel U.S. Toll-Free Line

Use the Actel toll-free line to contact Actel for sales information, technical support, requests for literature about Actel and Actel products, Customer Service, investor information, and using the Action Facts service.

The Actel Toll-Free Line is (888) 99-ACTEL.

## Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From Northeast and North Central U.S.A., call (408) 522-4480.
From Southeast and Southwest U.S.A., call (408) 522-4480.
From South Central U.S.A., call (408) 522-4434.
From Northwest U.S.A., call (408) 522-4434.
From Canada, call (408) 522-4480.
From Europe, call (408) 522-4252 or +44 (0) 1256 305600.
From Japan, call (408) 522-4743.
From the rest of the world, call (408) 522-4743.
Fax, from anywhere in the world (408) 522-8044.

## Customer Applications Center

The Customer Applications Center is staffed by applications engineers who can answer your hardware, software, and design questions.

All calls are answered by our Technical Message Center. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:30 a.m. to 5 p.m., Pacific Standard Time, Monday through Friday.

The Customer Applications Center number is (800) 262-1060.

European customers can call +44 (0) 1256 305600.

## Guru Automated Technical Support

Guru is a Web based automated technical support system accessible through the Actel home page (**http://www.actel.com/guru/**). Guru provides answers to technical questions about Actel products. Many answers include diagrams, illustrations and links to other resources on the Actel Web site. Guru is available 24 hours a day, seven days a week.

## Web Site

Actel has a World Wide Web home page where you can browse a variety of technical and non-technical information. Use a Net browser (Netscape recommended) to access Actel's home page.

The URL is **http://www.actel.com**. You are welcome to share the resources we have provided on the net.

Be sure to visit the "Actel User Area" on our Web site, which contains information regarding: products, technical services, current manuals, and release notes.

## FTP Site

Actel has an anonymous FTP site located at **ftp://ftp.actel.com**. You can directly obtain library updates, software patches, design files, and data sheets.

## Electronic Mail

You can communicate your technical questions to our e-mail address and receive answers back by e-mail, fax, or phone. Also, if you have design problems, you can e-mail your design files to receive assistance. The e-mail account is monitored several times per day.

The technical support e-mail address is **tech@actel.com**.

# Worldwide Sales Offices

## Headquarters

Actel Corporation
955 East Arques Avenue
Sunnyvale, California 94086
Toll Free: 888.99.ACTEL

Tel: 408.739.1010
Fax: 408.739.1540

## US Sales Offices

**California**

Bay Area
Tel: 408.328.2200
Fax: 408.328.2358

Irvine
Tel: 949.727.0470
Fax: 949.727.0476

San Diego
Tel: 619.938.9860
Fax: 619.938.9887

Thousand Oaks
Tel: 805.375.5769
Fax: 805.375.5749

**Colorado**

Tel: 303.420.4335
Fax: 303.420.4336

**Florida**

Tel: 407.677.6661
Fax: 407.677.1030

**Georgia**

Tel: 770.831.9090
Fax: 770.831.0055

**Illinois**

Tel: 847.259.1501
Fax: 847.259.1572

**Maryland**

Tel: 410.381.3289
Fax: 410.290.3291

**Massachusetts**

Tel: 978.244.3800
Fax: 978.244.3820

**Minnesota**

Tel: 612.854.8162
Fax: 612.854.8120

**North Carolina**

Tel: 919.376.5419
Fax: 919.376.5421

**Pennsylvania**

Tel: 215.830.1458
Fax: 215.706.0680

**Texas**

Tel: 972.235.8944
Fax: 972.235.965

## International Sales Offices

**Canada**
Suite 203
135 Michael Cowpland Dr,
Kanata, Ontario K2M 2E9

Tel: 613.591.2074
Fax: 613.591.0348

**France**
361 Avenue General de Gaulle
92147 Clamart Cedex

Tel: +33 (0)1.40.83.11.00
Fax: +33 (0)1.40.94.11.04

**Germany**
Bahnhofstrasse 15
85375 Neufahrn

Tel: +49 (0)8165.9584.0
Fax: +49 (0)8165.9584.1

**Hong Kong**
Suite 2206,
Parkside Pacific Place,
88 Queensway

Tel: +011.852.2877.6226
Fax: +011.852.2918.9693

**Italy**
Via Giovanni da Udine No. 34
20156 Milano

Tel: +39 (0)2.3809.3259
Fax: +39 (0)2.3809.3260

**Japan**
EXOS Ebisu Building 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150

Tel: +81 (0)3.3445.7671
Fax: +81 (0)3.3445.7668

**Korea**
135-090, 18th Floor,
Kyoung Am Building
157-27 Samsung-dong
Kangnam-ku, Seoul

Tel: +82 (0)2.555.7425
Fax: +82 (0)2.555.5779

**Taiwan**
4F-3, No. 75, Sec. 1,
Hsin-Tai-Wu Road,
Hsi-chih, Taipei, 221

Tel: +886 (0)2.698.2525
Fax: +886 (0)2.698.2548

**United Kingdom**
Daneshill House,
Lutyens Close
Basingstoke,
Hampshire RG24 8AG

Tel: +44 (0)1256.305600
Fax: +44 (0)1256.355420

# Index