



## Using Select-RAM Memory in XC4000 Series FPGAs

XAPP 057 July 7,1996 (Version 1.0)

Application Note by Lois Cartier

### Summary

XC4000-Series FPGAs include Select-RAM™ memory, which can be configured as ROM or as single- or dual-port RAM, with edge-triggered or level-sensitive timing. This application note describes how to implement Select-RAM memory in a design: in schematic entry, MemGen memory block generator, X-BLOX™ schematic-based synthesis, and HDL-synthesis environments. Specifying timing requirements, evaluating performance, and floorplanning are also described.

### Xilinx Family

XC4000E, XC4000EX, XC4000L, XC4000XL

### Demonstrates

Select-RAM memory

## Select-RAM Memory

Select-RAM memory is a major feature distinguishing Xilinx FPGAs from competitive devices. Select-RAM memory can be described as the capability of programming the look-up tables in XC4000-Series Configurable Logic Blocks (CLBs) as ROM or as single- or dual-port RAM, with edge-triggered or level-sensitive timing. The dual-port option considerably simplifies the implementation of some applications, such as FIFOs, while the simultaneous read/write functionality doubles the effective through-put. Edge-triggered (synchronous) timing doubles the speed again, as described in the application note "Implementing FIFOs in XC4000 Series RAM." The simplified timing also considerably accelerates the entire RAM design process. The application note "XC4000 Series Edge-Triggered and Dual-Port RAM Capability" provides a brief overview of Select-RAM functionality.

**Note:** At the time of this publication, software for the XC4000EX family was still under development. This application note is therefore directed to the 5.2.1/6.0.1 release of XACTstep™ (and the V1.0.0 XC4000E Pre-Release software, which is obsoleted by XACTstep V5.2.1/6.0.1). For specific instructions on using newer software releases or XC4000EX software with Select-RAM memory, see the documentation supplied with the software package. Most sections of this application note are independent of the Xilinx software release.

## Specifying Select-RAM Memory

Select-RAM memory can be used in any design environment that allows translation to Xilinx XC4000-Series FPGAs. The most common design entry methods include:

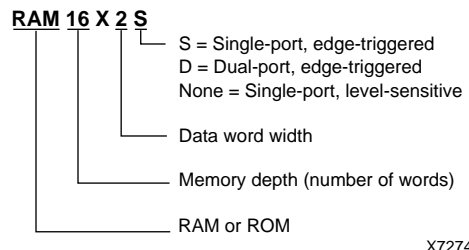
- Schematic entry
- MemGen — memory block generator from Xilinx
- X-BLOX — schematic-based synthesis from Xilinx
- HDL synthesis.

Use of Select-RAM memory in each of these design environments is described in this section.

For detailed information about the various configurations of Select-RAM memory, see the *XC4000 Series Field Programmable Gate Arrays* product specification and the application note "XC4000 Series Edge-Triggered and Dual-Port RAM Capability."

### Schematic Entry

When entering a design using a schematic editor, the selection of Select-RAM memory mode is made by placing the appropriate library symbol. The library symbol names are coded based on the size of the memory and the configuration mode, as shown in **Figure 1**. **Table 1** shows all memory blocks available in the Xilinx XC4000E and XC4000EX libraries. Each block is implemented as either a library primitive (basic component) or a library macro (created from library primitives and/or other library macros). Primitives and macros have different properties, as discussed throughout this application note.



X7274

**Figure 1: RAM Library Symbol Naming Conventions**

**Table 1: Select-RAM Symbols in Xilinx Libraries**

Symbol Name	Primitive or Macro	# of CLBs
RAM16X1	Primitive	0.5
RAM16X1D	Primitive	1
RAM16X1S	Primitive	0.5
RAM16X2	Macro	1
RAM16X2D	Macro	2
RAM16X2S	Macro	1
RAM16X4	Macro	2
RAM16X4D	Macro	4
RAM16X4S	Macro	2
RAM16X8	Macro	4
RAM16X8D	Macro	8
RAM16X8S	Macro	4
RAM32X1	Primitive	1
RAM32X1S	Primitive	1
RAM32X2	Macro	2
RAM32X2S	Macro	2
RAM32X4	Macro	4
RAM32X4S	Macro	4
RAM32X8	Macro	8
RAM32X8S	Macro	8
ROM16X1	Primitive	0.5
ROM32X1	Primitive	1

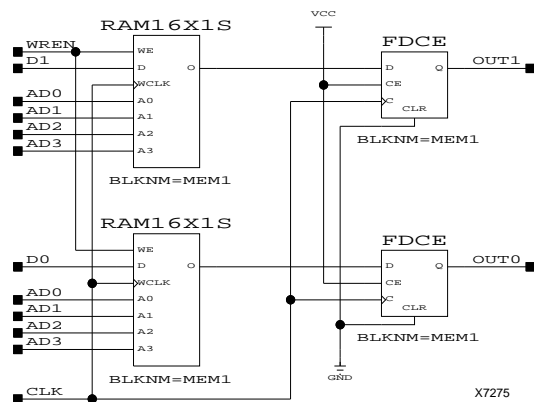
### Initializing Select-RAM in a Schematic

An initial value can be specified for Select-RAM blocks by attaching an INIT attribute or property to any RAM or ROM primitive. For example, the attribute "INIT=xxxx" can be attached to a placement of a RAM16X1S, RAM16X1D, RAM16X1, or ROM16X1 symbol, where xxxx is any 4-digit hexadecimal number. Any 8-digit hexadecimal number can be similarly attached to any of the 32-bit memory primitives.

The INIT attribute or property cannot be used on macros. To initialize a RAM macro, copy the macro to a new name, and attach INIT attributes to the primitives inside the macro.

The initialization values are specified beginning with the most significant number, e.g., INIT=020A places a "1" in bit locations 1, 3, and 9, and "0" at all other locations. The default initialization value for RAMs is all zeros.

The initial value is assigned to the memory at configuration only. Global Set Reset (GSR) has no effect on the memory contents.

**Figure 2: RAM Mapping Example Using BLKNM**

### Mapping Into CLBs

The XACTstep™ software automatically maps RAMs, ROMs, and flip-flops into Configurable Logic Blocks (CLBs). However, if desired, the library symbols designated as primitives in Table 1 can be marked with a BLKNM or HBLKNM attribute or property to group them with one or two flip-flops into a single CLB, as shown in Figure 2.

BLKNM and HBLKNM attributes cannot be used with macros. See the *Libraries Guide* for more information about these attributes.

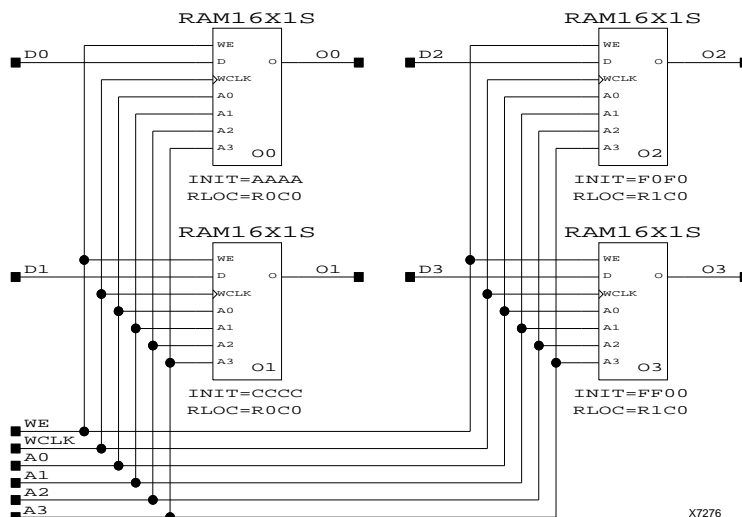
### Directing Placement of Memory Blocks from the Schematic

For larger memory blocks, substantially improved performance results if the RAM or ROM blocks are placed in patterns that minimize delay on address lines. (See "Floorplanning Select-RAM Memory" on page 15 for a discussion of optimal placement.) This placement can be specified either with the Xilinx Floorplanner, or directly in the schematic.

Placement of Select-RAM memory blocks can be controlled in the schematic by adding either location (LOC) or relative location (RLOC) attributes to the library symbols.

### LOC Attributes or Properties

LOC attributes specify a particular CLB or group of CLBs into which the RAM must be placed; therefore, it is not easy to change the location of a large memory block once it is placed using LOC attributes. Further, a macro locked to a particular location can only be used once in a design.



**Figure 3: 16 x 4 Single-Port Edge-Triggered RAM Schematic**

LOC attributes can be used with both library primitives and macros. For example, LOC=CLB\_R1C1 attached to a RAM16X2S symbol specifies that the 16x2 RAM block be placed in the CLB in the upper left corner of the device. For a symbol representing more than one CLB, such as a large library macro, a range of locations, such as LOC=CLB\_R1C1:CLB\_R4C4, can be specified. All memory blocks in the macro so tagged will be placed somewhere in the upper left 4 x 4 array of CLBs. To specify that all RAM or ROM in a macro be placed in the leftmost column of CLBs, specify LOC=CLB\_R\*C1.

### RLOC Attributes or Properties

RLOC attributes placed on two or more library symbols in a single schematic specify a relative location between the blocks. Macros with a given size and shape imposed by RLOC attributes can be automatically placed anywhere in a device, and moved by the software. They can, therefore, be used multiple times in one or more designs. The Relationally Placed Macros (RPMs) in the Xilinx library are implemented using RLOCs.

Figure 3 shows an example of a 16 x 4 single-port edge-triggered RAM constrained with RLOC attributes to be placed in two adjacent CLBs in a column. This simple example implements a customized version of the RAM16X4S macro from the Xilinx library. It is referenced throughout this application note.

For further information on LOC and RLOC attributes or properties, see the *Libraries Guide*.

### MemGen Memory Block Generator

The Xilinx MemGen program, called by the Xilinx Memory Generator, automatically generates RAM and ROM blocks, with accompanying control logic, according to specifications provided by the user.

One of the specifications required by MemGen is the RAM or ROM type. Type SYNC\_RAM specifies single-port edge-triggered (synchronous) RAM. Type DP\_RAM specifies dual-port RAM, which is always edge-triggered. Type RAM selects traditional, single-port level-sensitive RAM, and type ROM indicates that a ROM block (look-up table) should be generated.

A sample MemGen input file, implementing a 16 x 4 single-port edge-triggered RAM, is shown in Figure 4. Alternatively, MemGen can be run in interactive mode.

MemGen initialization data is specified one word at a time, with the data for address zero specified first. The initialization data in Figure 4 places a "0" at address 0000, a "1" at address 0001, etc. This file implements the same RAM, with the same initialization data, as the schematic shown in Figure 3. (However, the relative locations of the RAM blocks are not set in the MemGen output file.)

MemGen also creates a symbol that can be placed in a schematic, using Viewlogic or OrCAD schematic entry. The Foundation software Memory Generator creates a symbol for Aldec schematic entry. If these symbols are used in schematics, the XACT<sup>step</sup> software automatically includes the generated memory block in the netlist description.

MemGen is described in detail in the *Development System Reference Guide*.

```

; =====
; Memory file for 16 x 4 Single-Port Edge-Triggered RAM
; =====
TYPE SYNC_RAM                ; The memory is a Synchronous RAM Block
DEPTH 16                     ; The memory is 16 words deep
WIDTH 4                       ; Each memory word is 4 bits wide
PART 4005EPG156              ;
SYMBOL VIEWLOGIC PINS        ; Build a Viewlogic symbol with pin inputs (rather than bus inputs)
DEFAULT 0                     ; Use this default value for unspecified locations
DATA 0 1 2 3 4 5 6 7 8 9 a b c d e f ; Initialize RAM with this data

```

Figure 4: Sample MemGen Input File

## X-BLOX Schematic-Based Synthesis

X-BLOX is a set of parameterized library symbols and software that synthesizes these symbols into logic optimized for Xilinx FPGAs.

X-BLOX has three parameterized modules for implementing static RAMs: SYNC\_RAM, DP\_RAM, and SRAM. The SYNC\_RAM module implements single-port edge-triggered (synchronous) RAM; DP\_RAM is used for dual-port RAM, which is always edge-triggered, and SRAM implements level-sensitive (asynchronous) static RAM. Read-only memories are implemented with the PROM module. The library symbols for these modules are shown in Figure 5.

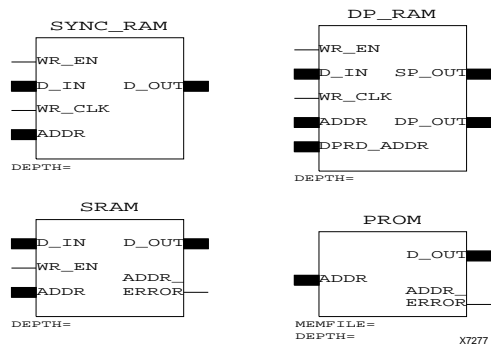


Figure 5: X-BLOX Select-RAM Library Symbols

To include an X-BLOX memory block in a schematic, simply place the appropriate symbol and set the necessary parameters by adding attributes or properties to the symbol. (The special X-BLOX library must be on the library search path for the schematic tool.) Symbol inputs, outputs, and supported attributes for the memory modules are shown in Table 2.

Figure 6 shows the X-BLOX implementation of a 16 x 4 single-port edge-triggered RAM block. The initialization file for this design is shown in Figure 7. The data format is the same as for the MemGen program. The initialization data in Figure 7 places a "0" at address 0000, a "1" at address 0001, etc. If initial values are not specified, the RAM is initialized to all zeros.

X-BLOX is fully described in the *X-BLOX Reference/User Guide*. Additional information on the Select-RAM memory blocks is available in the "Release Document, XACTstep V5.2.1/6.0.1".

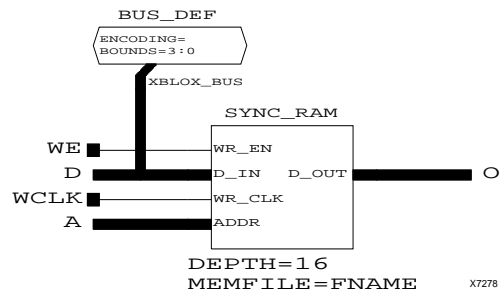


Figure 6: 16 x 4 Single-Port Edge-Triggered RAM Implemented in X-BLOX

```

; =====
; Initialization file for 16 x 4 RAM in X-BLOX
; =====
DATA 0 1 2 3 4 5 6 7 8 9 a b c d e f

```

Figure 7: Initialization File for X-BLOX RAM

**Table 2: X-BLOX Select-RAM Symbols: Inputs, Outputs, and Supported Attributes**

			SYNC_RAM	DP_RAM	SRAM	PROM
Inputs	Bus?					
ADDR	Yes	Address port	X	X	X	X
D_IN	Yes	Data Input port	X	X	X	
DPRD_ADDR	Yes	Dual-Port Read Address port		X		
WR_CLK	No	Load D_IN into RAM	X	X		
WR_EN	No	Write Enable, active-High	X	X	X	
Outputs	Bus?					
ADDR_ERROR	No	Address Error (out-of-bounds), active High			X	X
D_OUT	Yes	Data Output port	X		X	X
DP_OUT	Yes	Dual-Port Out, selected by DPRD_ADDR		X		
SP_OUT	Yes	Single-Port Out, selected by ADDR		X		
Supported Attributes						
BOUNDS		Defines width of the buses	X	X	X	X
DEPTH		Number of locations in the RAM	X	X	X	X
ENCODING		Bus data types	X	X	X	X
INIT		Specify initial RAM contents	X	X	X	X
MEMFILE		Specify name of file with initial RAM contents	X	X	X	X
TNM		Identify timing groups for XACT-Performance	X	X	X	X

## HDL Synthesis

Select-RAM memory elements are supported in most synthesis packages—including Synopsys, Cadence, Metamor, and Exemplar—through instantiation in the HDL source code. Although RAM can be described behaviorally, this methodology currently synthesizes to inefficient latch- or register-based implementations. ROM can be described behaviorally or instantiated, as desired.

MemGen blocks can be generated, as described in “[MemGen Memory Block Generator](#)” on [page 3](#), and then instan-

tiated in the code. Alternatively, the seven Select-RAM primitive library components from [Table 1](#) can be instantiated directly. [Table 3](#) shows the input and output port names for each of these elements.

[Figure 8](#) shows how to instantiate a MemGen module in VHDL code. The sample module is the 16 x 4 single-port edge-triggered RAM block shown in [Figure 3](#), [Figure 4](#), and [Figure 6](#).

**Table 3: Select-RAM Memory Components for HDL Synthesis**

Cell	Outputs	Inputs
<b>Single-Port, Edge-Triggered RAM</b>		
MemGen Output	O(n:o)	D(n:o), A(n:o), WE, WCLK
RAM16X1S	O	D, A3, A2, A1, A0, WE, WCLK
RAM32X1S	O	D, A4, A3, A2, A1, A0, WE, WCLK
<b>Dual-Port, Edge-Triggered RAM</b>		
MemGen Output	SPO(n:o), DPO(n:o)	D(n:o), A(n:o), DPRA(n:o), WE, WCLK
RAM16X1D	SPO, DPO	D, A3, A2, A1, A0, DPRA3, DPRA2, DPRA1, DPRA0, WE, WCLK
<b>Level-Sensitive RAM</b>		
MemGen Output	O(n:o)	D(n:o), A(n:o), WE
RAM16X1	O	D, A3, A2, A1, A0, WE
RAM32X1	O	D, A4, A3, A2, A1, A0, WE
<b>ROM</b>		
MemGen Output	O(n:o)	A(n:o)
ROM16X1	O	A3, A2, A1, A0
ROM32X1	O	A4, A3, A2, A1, A0

```

-- =====
-- Instantiating a Memgen 16 x 4 Single-Port Edge-Triggered RAM Module
-- =====

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity RAM_MEMGEN is
    port ( DATA: in STD_LOGIC_VECTOR (3 downto 0);
          ADDR: in STD_LOGIC_VECTOR (3 downto 0);
          WR_EN, WR_CLK: in STD_LOGIC;
          DOUT: out STD_LOGIC_VECTOR (3 downto 0) );
end RAM_MEMGEN;

architecture BEHAV of RAM_MEMGEN is
    component RAMDATA
        port ( D3, D2, D1, D0: in STD_LOGIC;
              A3, A2, A1, A0: in STD_LOGIC;
              WE, WCLK: in STD_LOGIC;
              O3, O2, O1, O0: out STD_LOGIC);
    end component;
begin
    u1: RAMDATA port map ( D3=>DATA(3),D2=>DATA(2),D1=>DATA(1),D0=>DATA(0),
                          A3=>ADDR(3),A2=>ADDR(2),A1=>ADDR(1),A0=>ADDR(0),
                          WE=>WR_EN,WCLK=>WR_CLK,
                          O3=>DOUT(3),O2=>DOUT(2),O1=>DOUT(1),O0=>DOUT(0) );

end BEHAV;

```

Figure 8: Instantiating a MemGen Module (VHDL)

```

// =====
// Instantiating a Memgen 16 x 4 Single-Port Edge-Triggered RAM Module
// =====

module RAM_MEMGEN (DATA, ADDR, WR_EN, WR_CLK, DOUT);
input [3:0] DATA, ADDR;
input WR_EN, WR_CLK;
output [3:0] DOUT;

RAMDATA U1(.D3(DATA[3]), .D2(DATA[2]), .D1(DATA[1]), .D0(DATA[0]),
           .A3(ADDR[3]), .A2(ADDR[2]), .A1(ADDR[1]), .A0(ADDR[0]),
           .WE(WR_EN), .WCLK(WR_CLK),
           .O3(DOUT[3]), .O2(DOUT[2]), .O1(DOUT[1]), .O0(DOUT[0])) ;

endmodule

module RAMDATA (D3, D2, D1, D0, A3, A2, A1, A0, WE, WCLK, O3, O2, O1, O0);
input D3, D2, D1, D0, A3, A2, A1, A0, WE, WCLK;
output O3, O2, O1, O0;

endmodule

```

Figure 9: Instantiating a MemGen Module (Verilog)

Figure 9 shows how to instantiate the same module in Verilog code. When instantiating a MemGen or other XNF format file in Verilog code, use the Synopsys “remove\_design” command on the module before writing out the SXNF file. (If this step is not performed, a “shell” of an SXNF file is written out, the Xilinx XNFMerge program reads this “shell” instead of the XNF file for the instantiated block, and XNFMerge does not complete.) When instantiating Unified Library primitives, or when using VHDL code, this step is not required.

Figure 10 shows the instantiation of four 16 x 1 library primitives in VHDL code, to implement the same 16 x 4 RAM block. Figure 11 also implements the same RAM function, but in Verilog format.

Additional information on implementing Select-RAM memory in VHDL can be found in *Synopsys (XSI) for FPGAs Interface/Tutorial Guide*, and in the “Release Document, XACTstep V5.2.1/6.0.1”.

```
-- =====
--   Instantiating Library Primitives to Create a 16 x 4 Single-Port Edge-Triggered RAM
-- =====
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity RAM_PRIM is
    port ( DATA : in STD_LOGIC_VECTOR (3 downto 0);
          ADDR : in STD_LOGIC_VECTOR (3 downto 0);
          WR_EN, WR_CLK : in STD_LOGIC;
          DOUT : out STD_LOGIC_VECTOR (3 downto 0) );
end RAM_PRIM;

architecture BEHAV of RAM_PRIM is

    component RAM16X1S
        port ( D, A3, A2, A1, A0, WE, WCLK : in STD_LOGIC;
              O : out STD_LOGIC );
    end component;

begin
    RAMCELL : RAM16X1S port map ( D=>DATA(0), A3=>ADDR(3), A2=>ADDR(2), A1=>ADDR(1),
                                  A0=>ADDR(0), WE=>WR_EN, WCLK=>WR_CLK, O=>DOUT(0) );
    RAMCELL_1: RAM16X1S port map ( D=>DATA(1), A3=>ADDR(3), A2=>ADDR(2), A1=>ADDR(1),
                                  A0=>ADDR(0), WE=>WR_EN, WCLK=>WR_CLK, O=>DOUT(1) );
    RAMCELL_2: RAM16X1S port map ( D=>DATA(2), A3=>ADDR(3), A2=>ADDR(2), A1=>ADDR(1),
                                  A0=>ADDR(0), WE=>WR_EN, WCLK=>WR_CLK, O=>DOUT(2) );
    RAMCELL_3: RAM16X1S port map ( D=>DATA(3), A3=>ADDR(3), A2=>ADDR(2), A1=>ADDR(1),
                                  A0=>ADDR(0), WE=>WR_EN, WCLK=>WR_CLK, O=>DOUT(3) );
end BEHAV;
```

**Figure 10: Instantiating Select-RAM Library Primitives (VHDL)**

```
// =====
//  Instantiating Library Primitives to Create a 16 x 4 Single-Port Edge-Triggered RAM
// =====

module RAM_PRIM (DATA, ADDR, WR_EN, WR_CLK, DOUT);
input [3:0] DATA;
input [3:0] ADDR ;
input WR_EN, WR_CLK;
output [3:0] DOUT ;
RAM16X1S RAMCELL (.D(DATA[0]), .A3(ADDR[3]), .A2(ADDR[2]), .A1(ADDR[1]), .A0(ADDR[0]),
    .WE(WR_EN), .WCLK(WR_CLK), .O(DOUT[0]) ) ;
RAM16X1S RAMCELL_1 (.D(DATA[1]), .A3(ADDR[3]), .A2(ADDR[2]), .A1(ADDR[1]), .A0(ADDR[0]),
    .WE(WR_EN), .WCLK(WR_CLK), .O(DOUT[1]) ) ;
RAM16X1S RAMCELL_2 (.D(DATA[2]), .A3(ADDR[3]), .A2(ADDR[2]), .A1(ADDR[1]), .A0(ADDR[0]),
    .WE(WR_EN), .WCLK(WR_CLK), .O(DOUT[2]) ) ;
RAM16X1S RAMCELL_3 (.D(DATA[3]), .A3(ADDR[3]), .A2(ADDR[2]), .A1(ADDR[1]), .A0(ADDR[0]),
    .WE(WR_EN), .WCLK(WR_CLK), .O(DOUT[3]) ) ;
endmodule
```

**Figure 11: Instantiating Select-RAM Library Primitives (Verilog)**

### Initializing Select-RAM Memory in Synthesized Designs

If instantiating MemGen blocks in the HDL code, an initial value can be specified by placing initial RAM values in the MemGen file. Because the MemGen module is a “black box” instantiation, the synthesis tool and the behavioral simulator are unaware of the contents of the module. Timing simulation after implementing and back-annotating the design is therefore the only means of simulating the contents of memory blocks generated by MemGen. This process conveys the initialization values to the simulator.

If instantiating library primitives directly in the code, the method for initializing RAM or ROM may differ depending on the synthesis tool. For Synopsys designs, enter initialization values for the RAM or ROM using the `set_attribute` command during synthesis. The procedure is the same for VHDL or Verilog code. For 16-location memory blocks, specify a 4-digit hexadecimal value. For 32-location memory blocks, specify an 8-digit hexadecimal value. An example of the `set_attribute` statements for the 16 x 4 single-port edge-triggered RAM example is included in [Figure 12](#). This file places a “0” at address 0000, a “1” at address 0001, etc. For instructions on initializing RAM or ROM when using other synthesis tools, see the user guide provided by the software company.

The initial value is assigned to the memory at configuration only. Global Set Reset (GSR) has no effect on the memory contents.

```
set_attribute RAMCELL xnf_init AAAA -type string
set_attribute RAMCELL_1 xnf_init CCCC -type string
set_attribute RAMCELL_2 xnf_init F0F0 -type string
set_attribute RAMCELL_3 xnf_init FF00 -type string
```

**Figure 12: Initializing Select-RAM Memory in Synopsys**

### Directing Placement of Synthesized Memory Blocks

For larger memory blocks, substantially improved performance results if the RAM or ROM blocks are placed in patterns that minimize delay on address lines. (See [“Floorplanning Select-RAM Memory” on page 15](#) for a discussion of optimal placement.) This placement can be specified with the Xilinx Floorplanner. Floorplanning techniques for synthesized Select-RAM blocks are discussed at length in the *HDL Synthesis for FPGAs Design Guide*.

## XDE Graphical Editor Support

Select-RAM memory is also supported in XDE, the Xilinx graphical Design Editor, for designers who require complete control over the implementation of their designs. For details on how to configure Select-RAM memory in XDE, see the “Release Document, XACTstep V5.2.1/6.0.1”.



## Select-RAM Memory Performance

XACT-Performance™ is a Xilinx software utility that enforces timing requirements on the implementation software. These specifications are taken into account during each phase of the design implementation process: mapping, placement, and routing. The designer can dictate required performance for a design containing Select-RAM memory with XACT-Performance, then use the Xilinx Timing Analyzer and/or industry-standard simulators on the implemented design to verify functionality at the required speed.

### Specifying Performance Requirements

Timing can be specified for paths through Select-RAM memory, as for any other paths in XC4000-Series designs.

#### Paths, Endpoints, and Groups

XACT-Performance specifications are dictated by identifying specific paths through the design, and tagging each path with the required timing for the path. Paths are defined by the two endpoints. Endpoints are specified by groups (i.e., from one group of elements to another group of elements). Groups can be pre-defined by Xilinx or defined by the user.

There are four pre-defined logic groups: FFS, LATCHES, PADS, and RAMS. The RAMS group refers to XC4000-Series Select-RAM blocks configured as RAM. When a performance requirement of the form:

FROM:RAMS:TO:*user\_group*=*max\_delay*      or  
FROM:*user\_group*:TO:RAMS=*max\_delay*

is used, only paths with RAM blocks at the source or destination are referenced. (ROM blocks cannot be made part of a group, as they are not clocked elements. They are treated as combinatorial logic.)

User-defined timegroups are created by using the TNM (timing name) attribute or property to flag each member of the group with the user-defined name of the group. To add all RAMs in a macro to the group *time\_group*, add a TNM attribute of the form:

TNM=RAMS:*time\_group*

to the symbol. Only the Select-RAM blocks within the macro are tagged as belonging to the group.

Groups defined with TNM attributes can be combined into larger groups using TIMEGRPs.

#### Three Types of Paths

Three types of paths can be identified.

- Paths with RAMs at the beginning (read cycle):  
Path timing includes worst-case delay from a change on D, WE, or WCLK to data valid.
- Paths with RAMs in the middle:  
These paths are traced through address pins only, since changes on D or WE are assumed to be of interest only when the RAM is being read during a write. To specify a path through another input, split the path into two segments — one ending at the RAM input pin, the other beginning at the RAM output pin — and specify the timing separately for each segment.
- Paths with RAMs at the end (write cycle):  
Path timing includes setup time at the destination pin. Paths are not traced to the DPRA address pins, as these pins define only a read address, and paths that end at a RAM are performing a write function.

Table 4 shows under what circumstances XACT-Performance specifications are applied to each type of path.

See the “XACT-Performance Utility” section of the *Development System Reference Guide* and the Interface Tutorials manuals from Xilinx for further information on defining groups, using groups to define paths, and specifying timing requirements for defined paths.

#### Three Methods of Entry

Timing specifications can be defined in any of three ways:

- Schematic entry
- Constraints files (enter manually, or generate from MakeTNM and AddTNM for HDL designs)
- Command-line options

The Interface Tutorials manuals describe at length methods for entering XACT-Performance specifications in schematics. An example using the Viewlogic schematic editor is shown in Figure 13. (Bus labels are omitted for clarity.)

**Table 4: When XACT-Performance Requirements Are Applied to Paths Containing Select-RAM Memory**

	RAM at the Beginning	RAM in the Middle	RAM at the End
<b>Single-Port Edge-Triggered</b>	Always applied	Applied to paths through address pins only (A[3:0] to O or A[4:0] to O)	Always applied
<b>Dual-Port Edge-Triggered</b>	Always applied	Applied to paths through address pins only (A[3:0] to SPO and DPRA[3:0] to DPO)	Applied to all paths except those ending at DPRA[3:0]
<b>Level-Sensitive</b>	Always applied	Applied to paths through address pins only (A[3:0] to O or A[4:0] to O)	Always applied

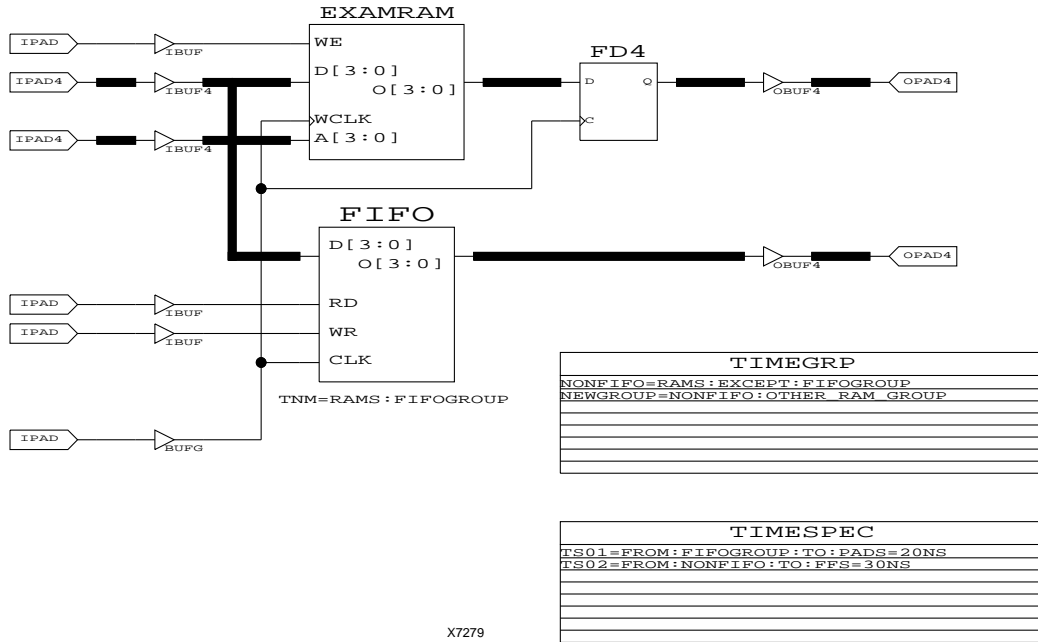


Figure 13: Schematic Specifying Timing Requirements (Viewlogic)

```
# =====
#                               Sample Constraints File for Edge-Triggered (Synchronous) RAM
# =====
# --- Defining new groups in the constraints file ---
# Define the group of all Select-RAM blocks labeled with a name beginning with "FIFO"
TIMEGRP = "fifogroup=RAMS(fifo*)";
# Define the group of all other Select-RAM blocks
TIMEGRP = "nonfifo=RAMS:EXCEPT:fifogroup";
# Combining groups defined in the constraints file, the schematic, or MakeTNM and AddTNM (synthesis)
TIMEGRP = "newgroup=nonfifo:other_ram_group";
#
# --- Specifying timing requirements using user-defined and pre-defined groups ---
# Read cycle timing: paths beginning at the RAM
TIMESPEC = "TS01=FROM:fifogroup:TO:PADS=20ns";
TIMESPEC = "TS02=FROM:nonfifo:TO:FFS=30ns";
# Paths through the RAM
TIMESPEC = "TS03=FROM:FFS:TO:FFS=20ns";
# Write cycle timing: paths ending at the RAM
TIMESPEC = "TS04=FROM:FFS:TO:fifogroup=20ns";
TIMESPEC = "TS05=FROM:FFS:TO:nonfifo=30ns";
```

Figure 14: Constraints File Specifying Timing for HDL Design

As an alternative to schematic entry, timing specifications can be entered using constraints files, as described in the *Libraries Guide*. **Figure 14** shows a sample constraints file defining several groups (TIMEGRP constraints) and performance requirements for these groups (TIMESPEC constraints).

A third method of entering XACT-Performance requirements is through PPR command-line options, as described in the “PPR” chapter of the *Development System Reference Guide*. However, this method does not allow as much flexibility as the other two alternatives. Level-sensitive RAM timing cannot be specified using this method. Edge-triggered timing can be broadly specified using the `dc2s=max_delay` option to define maximum clock-to-setup as in the following example:

```
ppr input_file dc2s=40
```

The clock-to-setup restriction is applied to all clocked elements in the design, including flip-flops, latches, and edge-triggered RAMs.

Any of these three methods can be used for schematic-based designs. Either constraints files or command-line options can be used for designs entered using HDL source code.

### Recommended Method for HDL Designs

The constraints file method of specifying timing requirements is recommended for HDL designs.

Synopsys has the capability of placing timing requirements directly in the Xilinx netlist (SXNF file), but the list of constraints so created is likely to be more extensive than is necessary or desirable. Instead, prevent the software from writing timing specifications to the SXNF file by using the “`xnfout_constraints_per_endpoint = 0`” command.

For HDL designs, use the MakeTNM and AddTNM programs to generate the constraints file. This procedure is described in detail in the *HDL Synthesis for FPGAs Design Guide*.

### Analyzing Select-RAM Performance

Xilinx also offers timing analysis of implemented designs. In the XACT<sub>step</sub> software, the Timing Analyzer (a shell for a program called XDelay) is a static timing analyzer that reports the worst-case timing delays of a routed FPGA design. This software is a sophisticated graphical tool fully capable of analyzing timing for Select-RAM designs.

The Timing Analyzer follows the same rules for tracing paths as are used by the XACT-Performance utility. (See **“Three Types of Paths” on page 9.**)

The Timing Analyzer can report three types of information.

- The Performance Summary reports worst-case timing for each of four typical design path types: pad-to-setup, clock-to-setup, clock-to-pad, and pad-to-pad.

- Performance To TimeSpecs reports which XACT-Performance constraints are met, and reports all constrained paths in detail.
- The Detailed Path report displays detailed path timing information for designated paths according to options set by the user.

In the Performance Summary report, edge-triggered RAM is included in the clock-to-setup report. Since edge-triggered RAM is clocked, it is essentially treated as if it were an addressable flip-flop.

For more information about the Timing Analyzer, see the *Timing Analyzer Reference/User Guide* and “The XDelay Timing Analysis Program” in the *Development System Reference Guide*. Tutorials demonstrating how to perform a simple timing analysis are included in the Interface Tutorial manuals.

### Simulating Select-RAM Memory

Industry-standard timing simulators are supported via simulation netlists that include timing information. Xilinx-supplied simulation models accurately model Select-RAM timing.

Simulation of edge-triggered RAM modes is much simpler than simulation of level-sensitive RAM. All that is necessary is to provide a clock at the specified frequency, and demonstrate that the read, write, and read-during-write cycles function at the desired clock speed without generating setup or hold violations.

Functional and timing simulation of Xilinx designs is described in the appropriate Interface Guides and demonstrated in the Interface Tutorial manuals.

### Simulating Select-RAM Memory in Viewlogic

A Viewlogic simulation command file is shown in **Figure 15**. This command file is directed to the 16 x 4 single-port edge-triggered RAM shown in **Figure 3** or generated from the MemGen file in **Figure 4**. The command file tests the initialization contents of the memory (which should be 0, 1, 2, 3, etc.). It then re-writes the RAMs with the inverse sequence (15, 14, 13, etc.), and verifies that the data is actually stored. Pattern files are used to supply input data and comparison data for the output, as shown in **Figure 16**.

Extra steps are required to include initial values in the simulation, when they are specified for a Select-RAM block.

The Viewlogic simulator does not automatically implement the INIT attribute on RAM or ROM symbols. The Loadm command must be used to enter the initialization value before simulating the design. The Loadm command can only be issued from the keyboard or from a command file; it does not appear in the menus.

The syntax for the Loadm command is as follows:

```
loadm instance_name\RAM [lowaddr:highaddr value\radix]
```

```
|=====|
|  Viewlogic Simulation Command File  |
|=====|

| roll back to time 0, reset internal network nodes
restart
| initialize RAM: call command file in Figure 17
execute examram.xmm

| define input and output vectors
vector A A3 A2 A1 A0
vector D D3 D2 D1 D0
vector O O3 O2 O1 O0
vector ram16x4s A3 A2 A1 A0 D3 D2 D1 D0 WE +
    WCLK O3 O2 O1 O0

| define waveforms and text output
wave examram.wfm A D WE WCLK O
watch A D WE WCLK O
break ram16x4s ? do (print > examram.out)

| define clock and initial values for inputs
clock WCLK 1 0
step 50ns
I WE
assign A 0\h
assign D F\h
cycle

| check initial contents of the RAM
| at each location, the output data should equal the
| address. First set the address, then, one clock
| cycle later, check the data
every 100ns for 16 do ( assign A < up1.pat )
after 100ns do +
    ( every 100ns for 16 do ( check O < up2.pat ) )
cycle 16

| test the ability to store new data in the RAM
| enable RAM write
s 50ns
h WE
s 50ns
| load f, e, d, ... 1, 0 into addresses 0, 1, ... e, f.
| verify data after each write
every 100ns for 16 do +
    ( assign A < up1.pat; assign D < down1.pat )
after 100ns do +
    ( every 100ns for 16 do ( check O < down2.pat ) )
cycle 16
```

Figure 15: Viewlogic Simulation Command File for 16 x 4 Single-Port Edge-Triggered RAM

up1.pat & up2.pat	down1.pat & down2.pat
0\h	f\h
1\h	e\h
2\h	d\h
3\h	c\h
4\h	b\h
5\h	a\h
6\h	9\h
7\h	8\h
8\h	7\h
9\h	6\h
a\h	5\h
b\h	4\h
c\h	3\h
d\h	2\h
e\h	1\h
f\h	0\h

Figure 16: Up and Down Pattern Files

```
|=====|
|  examram.xmm: Initialization Command File  |
|=====|

LOADM O0\RAM (15:15) 1\h
LOADM O0\RAM (14:14) 0\h
LOADM O0\RAM (13:13) 1\h
LOADM O0\RAM (12:12) 0\h
LOADM O0\RAM (11:11) 1\h
LOADM O0\RAM (10:10) 0\h
LOADM O0\RAM (9:9) 1\h
LOADM O0\RAM (8:8) 0\h
LOADM O0\RAM (7:7) 1\h
LOADM O0\RAM (6:6) 0\h
LOADM O0\RAM (5:5) 1\h
LOADM O0\RAM (4:4) 0\h
LOADM O0\RAM (3:3) 1\h
LOADM O0\RAM (2:2) 0\h
LOADM O0\RAM (1:1) 1\h
LOADM O0\RAM (0:0) 0\h
.
.
. (48 lines omitted)
```

Figure 17: Viewlogic Simulation Command File for Initializing 16 x 4 Single-Port Edge-Triggered RAM

At this time, *lowadrs* and *highadrs* must be equal; in other words, a separate Loadm statement is required for each bit. A sample command file containing Loadm statements is shown in [Figure 17](#). This command file is called by the simulation command file shown in [Figure 15](#).

Rather than tediously generating the Loadm statements by hand, they can be generated using either the XNF2WIR program or the XSimMake program, which calls XNF2WIR. XNF2WIR places the Loadm statements in a file with the extension XMM. (The -r option can be used to create the XMM file without generating a WIR file, if desired.) The XMM file can be referenced from a Viewlogic simulation command file, or the statements can be copied into the command file. XNF2WIR must be run on the top-level schematic, in order to preserve the naming hierarchy.

### Simulating Select-RAM Memory in Synopsys

For a Synopsys design, no special steps are required to perform timing simulation of Select-RAM memory. [Figure 18](#) shows an implementation of the familiar 16 x 4 single-port edge-triggered RAM design. A VHDL test bench for this file is shown in [Figure 19](#). The test bench tests the initialization contents of the memory (which should be 0, 1, 2, 3, etc.). It then re-writes the RAMs with the inverse sequence (15, 14, 13, etc.), and verifies that the data is actually stored.

[Figure 18](#) provides an example of a second method of instantiating memory primitives in a VHDL file. Using the “generate” statement, the synthesis software repeatedly instantiates the referenced component for each value of the index. This statement makes the creation of wide memory blocks more succinct.

For behavioral simulation, initial values for the RAM or ROM (other than the zero default value) can be specified in the HDL file using the “constant” statement, as shown in [Figure 18](#). It is vital to remember that *this initialization value is totally ignored during compilation*, along with everything between the *translate\_off* and *translate\_on* statements. In this example, *the behavioral simulation will not match the behavior of the compiled design, unless the initial values are also specified during compilation*.

Initial values are attached to RAM or ROM primitives during compilation by using the *set\_attribute* command, as described in “[Initializing Select-RAM Memory in Synthesized Designs](#)” on page 8. The commands shown in [Figure 12](#) can be used to initialize this example. If these commands are properly issued while compiling a design, the initial values are already present when performing a timing simulation on the back-annotated netlist.

<pre> library IEEE; use IEEE.STD_LOGIC_1164.all; library XC4000E; use XC4000E.COMPONENTS.all;  entity RAM is     port (ADDR : in STD_LOGIC_VECTOR (3 downto 0);           DATA_IN : in STD_LOGIC_VECTOR (3 downto 0);           DATA_OUT: out STD_LOGIC_VECTOR (3 downto 0));     WEN : in STD_LOGIC;     WCK : in STD_LOGIC; end RAM;  architecture TEST of RAM is      component RAM16X1S     -- synopsys translate_off     generic (INIT : string(4 downto 1) );     -- synopsys translate_on     port (D, A3, A2, A1, A0, D, WE, WCLK : in STD_LOGIC;           O : out STD_LOGIC);     end component;      type INIT_DATA_ARRAY is array(0 to 3) of string(4 downto 1);      -- (continued in next column) </pre>	<pre> -- These values are for behavioral simulation only. -- These values are IGNORED during compilation. -- Use "set_attribute" command during compilation -- to assign initial values to the synthesized design. constant INIT_VALUES : INIT_DATA_ARRAY :=     ("AAAA", "CCCC", "F0F0", "FF00");  begin      BUILDGRAM : for i in 0 to 3 generate         RAMCELL : RAM16X1S             -- synopsys translate_off             generic map ( init =&gt; INIT_VALUES(i) )             -- synopsys translate_on             port map (D=&gt;DATA_IN(i),                       O=&gt;DATA_OUT(i),                       A3=&gt;ADDR(3),                       A2=&gt;ADDR(2),                       A1=&gt;ADDR(1),                       A0=&gt;ADDR(0),                       WE=&gt;WEN,                       WCLK=&gt;WCK );         end generate BUILDGRAM;      end TEST; </pre>
--	--

**Figure 18: VHDL Code for 16 x 4 Single-Port Edge-Triggered RAM**

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

entity RAM_TESTBENCH is
end RAM_TESTBENCH;

architecture TEST of RAM_TESTBENCH is

component RAM
  port (ADDR      : in STD_LOGIC_VECTOR (3 downto 0);
        DATA_IN  : in STD_LOGIC_VECTOR (3 downto 0);
        DATA_OUT : out STD_LOGIC_VECTOR (3 downto 0));
        WEN       : in STD_LOGIC;
        WCK       : in STD_LOGIC;
end component;

signal COUNT      : STD_LOGIC_VECTOR (3 downto 0);
signal WRITE_DATA : STD_LOGIC_VECTOR (3 downto 0);
signal READ_DATA  : STD_LOGIC_VECTOR (3 downto 0);
signal WRITE_ENABLE : STD_LOGIC;
signal WRITE_CLOCK : STD_LOGIC;

begin

  UUT : RAM port map ( ADDR => COUNT,
                      DATA_IN => WRITE_DATA,
                      DATA_OUT => READ_DATA,
                      WEN => WRITE_ENABLE,
                      WCK => WRITE_CLOCK );

  DRIVER : process
  begin

    -- Test initial contents of the RAM
    WRITE_ENABLE <= '0';
    WRITE_CLOCK <= '0';

    -- (continued in next column)

    for i in 0 to 15 loop

      COUNT <= CONV_STD_LOGIC_VECTOR(i,4);
      wait for 50 NS;
      assert READ_DATA = COUNT
        report "Initialization data mismatch!"
        severity error;
      wait for 50 NS;

    end loop;

    -- Test the ability to store new data in the RAM
    WRITE_ENABLE <= '1';

    for i in 0 to 15 loop

      WRITE_CLOCK <= '0';
      COUNT <= CONV_STD_LOGIC_VECTOR(i,4);
      WRITE_DATA <=
        CONV_STD_LOGIC_VECTOR(15-i,4);
      wait for 50 NS;
      WRITE_CLOCK <= '1';
      wait for 50 NS;
      assert READ_DATA = WRITE_DATA
        report "RAM contents data mismatch!"
        severity error;

    end loop;

    wait;

  end process;

end TEST;

configuration CFG_tb of RAM_TESTBENCH is
for TEST
end for;
end CFG_tb;

```

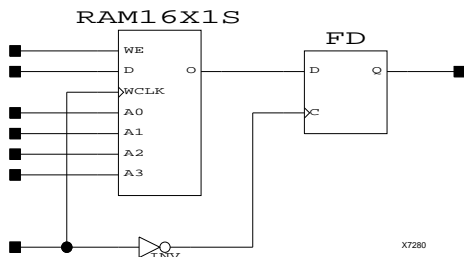
Figure 19: VHDL Test Bench for 16 x 4 Single-Port Edge-Triggered RAM

## Design Tip: Optimizing Performance by Using Both Clock Edges

One XC4000-Series CLB includes two function generators or look-up tables (LUTs) and two flip-flops. (In XC4000EX devices, each flip-flop can also be configured as a latch.) The LUTs can be configured as RAM, with the RAM output optionally registered in the same CLB.

For an edge-triggered RAM element, the RAM output can be clocked into the register on either the same clock edge used to write data, or the opposite edge. (See [Figure 20](#).) The Select-RAM was specifically designed to support either configuration, provided all timing specifications are followed.

The advantage of the configuration shown in [Figure 20](#) is that the read data is accessible half a clock cycle sooner than it would be if the same clock edge was used for writing and for registering the data.



**Figure 20: Registering RAM Data on Opposite Clock Edge**

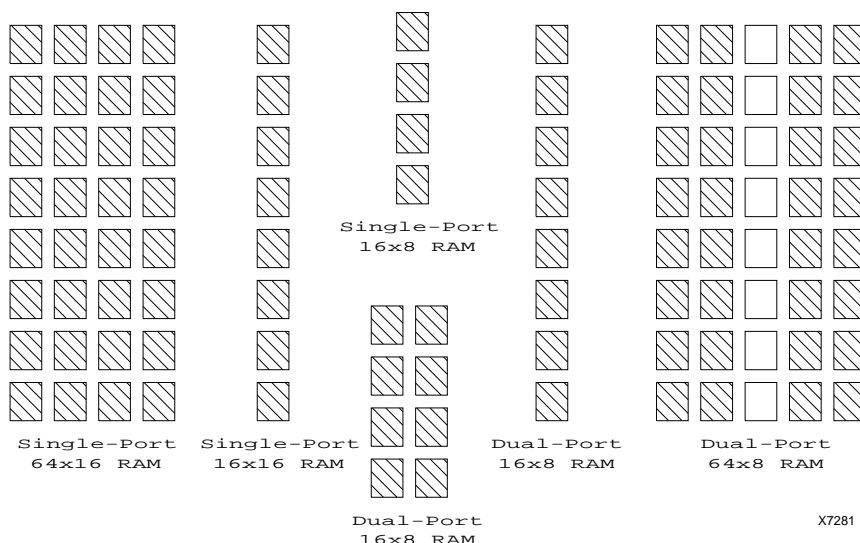
## Floorplanning Select-RAM Memory

For any memory block larger than a few CLBs, the address lines are typically the critical paths in the design, as they have the highest fanout. They are best routed on longlines or quad lines. Experience has shown that memory blocks using multiple CLBs are best placed in one or more vertical columns.

Dual-port RAM uses almost twice as many address lines as single-port RAM or ROM, for the same number of CLBs. For dual-port designs in the XC4000E, it is recommended that no more than two adjacent CLB columns be used for RAM. Place control logic in every third column to reduce potential vertical routing congestion. In the XC4000EX, with its additional routing channels, routing the address lines should not be a problem, even for large dual-port RAMs.

[Figure 21](#) shows efficient performance-optimized placements for single- and dual-port Select-RAM blocks in XC4000E devices.

The XACT<sup>step</sup> software provides three different methods of floorplanning: schematic RLOC and LOC attributes or properties, constraints files, and the Xilinx Floorplanner. RLOC attributes are particularly useful when creating a block (RPM) that will be reused in several schematics. Constraints files can be useful for synthesized designs, but the Xilinx Floorplanner is the recommended method for synthesized designs and for schematic macros not intended for multiple placements. The Xilinx Floorplanner is described in the *Floorplanner Reference/User Guide*. An on-line tutorial is also available on the PC platform.



**Figure 21: Efficient Select-RAM Block Placements**



## Additional Information

This application note covers a wide variety of subjects. References are made throughout the text to additional materials that can be accessed to obtain more detailed information on these subjects. These references are summarized below. All are Xilinx publications; some are available at the Xilinx WEB site at <http://www.xilinx.com>.

### Select-RAM Memory, General Information

- 1996 *Programmable Logic Data Book*: XC4000 Series Field Programmable Gate Arrays product specification
- "Implementing FIFOs in XC4000 Series RAM" (application note)
- "XC4000 Series Edge-Triggered and Dual-Port RAM Capability" (application note)

### Schematic Entry

- Interface User Guide for the appropriate interface
- *Libraries Guide* (attributes and properties)

### MemGen Memory Generator Tool

- *Development System Reference Guide*

### X-BLOX Schematic-Based Synthesis

- X-BLOX Reference/User Guide
- "Release Document, XACTstep V5.2.1/6.0.1" or
- "Release Document, XC4000E V1.0.0 Pre-Release"

### HDL Synthesis

- *Synopsys (XSI) for FPGAs Interface/Tutorial Guide*
- "Release Document, XACTstep V5.2.1/6.0.1" or
- "Release Document, XC4000E V1.0.0 Pre-Release"

### Floorplanning

- *HDL Synthesis for FPGAs Design Guide* (HDL)
- *Floorplanner Reference/User Guide*
- On-line tutorial (PC platform only)

### XDE Graphical Editor Support

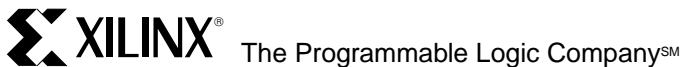
- "Release Document, XACTstep V5.2.1/6.0.1" or
- "Release Document, XC4000E V1.0.0 Pre-Release"

### XACT-Performance

- *Development System Reference Guide*: XACT-Performance Utility
- Interface Tutorial manual for the appropriate interface
- *Libraries Guide* (constraints files syntax)
- *HDL Synthesis for FPGAs Design Guide* (HDL designs)

### Timing Analyzer

- *Timing Analyzer Reference/User Guide*
- *Development System Reference Guide*: The XDelay Timing Analysis Program
- Interface Tutorial manual for the appropriate interface



#### Headquarters

Xilinx, Inc.  
2100 Logic Drive  
San Jose, CA 95124  
U.S.A.  
Tel: 1 (800) 255-7778  
or 1 (408) 559-7778  
Fax: 1 (800) 559-7114  
Net: [hotline@xilinx.com](mailto:hotline@xilinx.com)  
Web: <http://www.xilinx.com>

#### North America

Irvine, California  
(714) 727-0780

Englewood, Colorado  
(303) 220-7541

Sunnyvale, California  
(408) 245-9850

Schaumburg, Illinois  
(847) 605-1972

Nashua, New Hampshire  
(603) 891-1098

Raleigh, North Carolina  
(919) 846-3922

West Chester, Pennsylvania  
(610) 430-3300

Dallas, Texas  
(214) 960-1043

#### Europe

Xilinx Sarl  
Jouy en Josas, France  
Tel: (33) 1-34-63-01-01  
Net: [frhelp@xilinx.com](mailto:frhelp@xilinx.com)

Xilinx GmbH  
Aschheim, Germany  
Tel: (49) 89-99-1549-01  
Net: [dlhelp@xilinx.com](mailto:dlhelp@xilinx.com)

Xilinx, Ltd.  
Byfleet, United Kingdom  
Tel: (44) 1-932-349401  
Net: [ukhelp@xilinx.com](mailto:ukhelp@xilinx.com)

#### Japan

Xilinx, K.K.  
Tokyo, Japan  
Tel: (03) 3297-9191

#### Asia Pacific

Xilinx Asia Pacific  
Hong Kong  
Tel: (852) 2424-5200  
Net: [hongkong@xilinx.com](mailto:hongkong@xilinx.com)

© 1996 Xilinx, Inc. All rights reserved. The Xilinx name and the Xilinx logo are registered trademarks, all XC-designated products are trademarks, and the Programmable Logic Company is a service mark of Xilinx, Inc. All other trademarks and registered trademarks are the property of their respective owners.

Xilinx, Inc. does not assume any liability arising out of the application or use of any product described herein; nor does it convey any license under its patent, copyright or maskwork rights or any rights of others. Xilinx, Inc. reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx, Inc. cannot assume responsibility for the use of any circuitry described other than circuitry entirely embodied in its products. Products are manufactured under one or more of the following U.S. Patents: (4,847,612; 5,012,135; 4,967,107; 5,023,606; 4,940,909; 5,028,821; 4,870,302; 4,706,216; 4,758,985; 4,642,487; 4,695,740; 4,713,557; 4,750,155; 4,821,233; 4,746,822; 4,820,937; 4,783,607; 4,855,669; 5,047,710; 5,068,603; 4,855,619; 4,835,418; and 4,902,910. Xilinx, Inc. cannot assume responsibility for any circuits shown nor represent that they are free from patent infringement or of any other third party right. Xilinx, Inc. assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made.