Recommended reading

• Spartan-6 FPGA Configurable Logic Block: User Guide

  Google search: UG384

• Spartan-6 FPGA Block RAM Resources: User Guide

  Google search: UG383

• Xilinx FPGA Embedded Memory Advantages: White Paper

  Google search: WP360
Recommended reading

- XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices
  Chapter 7, HDL Coding Techniques
  Sections:
    - RAM HDL Coding Techniques
    - ROM HDL Coding Techniques
  - ISE In-Depth Tutorial, Section: Creating a CORE Generator Tool Module
Memory Types
Memory Types

Memory

ROM

RAM

Memory

Single port

Dual port

Memory

With asynchronous read

With synchronous read
Memory Types specific to Xilinx FPGAs

Memory

- Distributed (MLUT-based)
- Block RAM-based (BRAM-based)

Memory

- Inferred
- Instantiated

- Manually
- Using CORE Generator
FPGA Distributed Memory
Location of Distributed RAM

Graphics based on The Design Warrior’s Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043
Copyright © 2004 Mentor Graphics Corp. (www.mentor.com)
# Three Different Types of Slices in Spartan 6

<table>
<thead>
<tr>
<th>Feature</th>
<th>SLICEX</th>
<th>SLICEL</th>
<th>SLICEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-Input LUTs</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>8 Flip-flops</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Wide Multiplexers</td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Carry Logic</td>
<td>√</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Distributed RAM</td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>Shift Registers</td>
<td></td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>
Spartan-6 Multipurpose LUT (MLUT)

- 32-bit SR
- 64 x 1 RAM
- 64 x 1 ROM (logic)
Single-port 64 x 1-bit RAM

RAM64X1S

D
A[5:0]
WCLK
WE

O

Output
Registered Output
(Optional)

ug384_09_042309
Single-port 64 x 1-bit RAM

RAM64X1S

D
A[5:0]
WCLK
WE
(DX)
6 (D[6:1])
6 (CLK)
(WE/CE)

□ SPRAM64

DI1
O6
A[6:1]
WA[6:1]
CLK
WE

-> Output

 Registered Output (Optional)

ug384_09_042309
Memories Built of Neighboring MLUTs

Memories built of 2 MLUTs:

- Single-port 128 x 1-bit RAM: RAM128x1S
- Dual-port 64 x 1-bit RAM: RAM64x1D

Memories built of 4 MLUTs:

- Single-port 256 x 1-bit RAM: RAM256x1S
- Dual-port 128 x 1-bit RAM: RAM128x1D
- Quad-port 64 x 1-bit RAM: RAM64x1Q
- Simple-dual-port 64 x 3-bit RAM: RAM64x3SDP
  (one address for read, one address for write)
Dual-port 64 x 1 RAM

RAM64X1D

- D
- A[5:0]
- WCLK
- WE
- DPRA[5:0]

- SPO
- Registered Output (Optional)
- DPO
- Registered Output (Optional)
Dual-port 64 x 1 RAM

- Dual-port 64 x 1-bit RAM: 64x1D
- Single-port 128 x 1-bit RAM: 128x1S

Diagram shows the connectivity and signals for the RAM64X1D circuit, including inputs D, A[5:0], WCLK, WE, and outputs SPO, DQ, and DPO.
## Total Size of Distributed RAM

<table>
<thead>
<tr>
<th>Device</th>
<th>Logic Cells⁽¹⁾</th>
<th>Configurable Logic Blocks (CLBs)</th>
<th></th>
<th></th>
<th>Max Distributed RAM (Kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Slices⁽²⁾</td>
<td>Flip-Flops</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XC6SLX4</td>
<td>3,840</td>
<td>600</td>
<td>4,800</td>
<td></td>
<td>75</td>
</tr>
<tr>
<td>XC6SLX9</td>
<td>9,152</td>
<td>1,430</td>
<td>11,440</td>
<td></td>
<td>90</td>
</tr>
<tr>
<td>XC6SLX16</td>
<td>14,579</td>
<td>2,278</td>
<td>18,224</td>
<td></td>
<td>136</td>
</tr>
<tr>
<td>XC6SLX25</td>
<td>24,051</td>
<td>3,758</td>
<td>30,064</td>
<td></td>
<td>229</td>
</tr>
<tr>
<td>XC6SLX45</td>
<td>43,661</td>
<td>6,822</td>
<td>54,576</td>
<td></td>
<td>401</td>
</tr>
<tr>
<td>XC6SLX75</td>
<td>74,637</td>
<td>11,662</td>
<td>93,296</td>
<td></td>
<td>692</td>
</tr>
<tr>
<td>XC6SLX100</td>
<td>101,261</td>
<td>15,822</td>
<td>126,576</td>
<td></td>
<td>976</td>
</tr>
<tr>
<td>XC6SLX150</td>
<td>147,443</td>
<td>23,038</td>
<td>184,304</td>
<td></td>
<td>1,355</td>
</tr>
<tr>
<td>XC6SLX25T</td>
<td>24,051</td>
<td>3,758</td>
<td>30,064</td>
<td></td>
<td>229</td>
</tr>
<tr>
<td>XC6SLX45T</td>
<td>43,661</td>
<td>6,822</td>
<td>54,576</td>
<td></td>
<td>401</td>
</tr>
<tr>
<td>XC6SLX75T</td>
<td>74,637</td>
<td>11,662</td>
<td>93,296</td>
<td></td>
<td>692</td>
</tr>
<tr>
<td>XC6SLX100T</td>
<td>101,261</td>
<td>15,822</td>
<td>126,576</td>
<td></td>
<td>976</td>
</tr>
<tr>
<td>XC6SLX150T</td>
<td>147,443</td>
<td>23,038</td>
<td>184,304</td>
<td></td>
<td>1,355</td>
</tr>
</tbody>
</table>
FPGA Block RAM
Location of Block RAMs

Graphics based on The Design Warrior’s Guide to FPGAs
Devices, Tools, and Flows. ISBN 0750676043
Copyright © 2004 Mentor Graphics Corp. (www.mentor.com)
## Spartan-6 Block RAM Amounts

<table>
<thead>
<tr>
<th>Device</th>
<th>Logic Cells$^{(1)}$</th>
<th>Configurable Logic Blocks (CLBs)</th>
<th>DSP48A1 Slices$^{(3)}$</th>
<th>Block RAM Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Slices$^{(2)}$</td>
<td>Flip-Flops</td>
<td>Max Distributed RAM (Kb)</td>
</tr>
<tr>
<td>XC6SLX4</td>
<td>3,840</td>
<td>600</td>
<td>4,800</td>
<td>75</td>
</tr>
<tr>
<td>XC6SLX9</td>
<td>9,152</td>
<td>1,430</td>
<td>11,440</td>
<td>90</td>
</tr>
<tr>
<td>XC6SLX16</td>
<td>14,579</td>
<td>2,278</td>
<td>18,224</td>
<td>136</td>
</tr>
<tr>
<td>XC6SLX25</td>
<td>24,051</td>
<td>3,758</td>
<td>30,064</td>
<td>229</td>
</tr>
<tr>
<td>XC6SLX45</td>
<td>43,661</td>
<td>6,822</td>
<td>54,576</td>
<td>401</td>
</tr>
<tr>
<td>XC6SLX75</td>
<td>74,637</td>
<td>11,662</td>
<td>93,296</td>
<td>692</td>
</tr>
<tr>
<td>XC6SLX100</td>
<td>101,261</td>
<td>15,822</td>
<td>126,576</td>
<td>976</td>
</tr>
<tr>
<td>XC6SLX150</td>
<td>147,443</td>
<td>23,038</td>
<td>184,304</td>
<td>1,355</td>
</tr>
<tr>
<td>XC6SLX25T</td>
<td>24,051</td>
<td>3,758</td>
<td>30,064</td>
<td>229</td>
</tr>
<tr>
<td>XC6SLX45T</td>
<td>43,661</td>
<td>6,822</td>
<td>54,576</td>
<td>401</td>
</tr>
<tr>
<td>XC6SLX75T</td>
<td>74,637</td>
<td>11,662</td>
<td>93,296</td>
<td>692</td>
</tr>
<tr>
<td>XC6SLX100T</td>
<td>101,261</td>
<td>15,822</td>
<td>126,576</td>
<td>976</td>
</tr>
<tr>
<td>XC6SLX150T</td>
<td>147,443</td>
<td>23,038</td>
<td>184,304</td>
<td>1,355</td>
</tr>
</tbody>
</table>
Block RAM can have various configurations (port aspect ratios)
Parity Optional
(16K bits data, 2K bits parity)
## Block RAM Port Aspect Ratios

<table>
<thead>
<tr>
<th>Organization</th>
<th>Memory Depth</th>
<th>Data Width</th>
<th>Parity Width</th>
<th>DI/DO</th>
<th>DIP/DOP</th>
<th>ADDR</th>
<th>Total RAM Kbits</th>
</tr>
</thead>
<tbody>
<tr>
<td>512x36</td>
<td>512</td>
<td>32</td>
<td>4</td>
<td>(31:0)</td>
<td>(3:0)</td>
<td>(8:0)</td>
<td>18K</td>
</tr>
<tr>
<td>1Kx18</td>
<td>1024</td>
<td>16</td>
<td>2</td>
<td>(15:0)</td>
<td>(1:0)</td>
<td>(9:0)</td>
<td>18K</td>
</tr>
<tr>
<td>2Kx9</td>
<td>2048</td>
<td>8</td>
<td>1</td>
<td>(7:0)</td>
<td>(0:0)</td>
<td>(10:0)</td>
<td>18K</td>
</tr>
<tr>
<td>4Kx4</td>
<td>4096</td>
<td>4</td>
<td>-</td>
<td>(3:0)</td>
<td>-</td>
<td>(11:0)</td>
<td>16K</td>
</tr>
<tr>
<td>8Kx2</td>
<td>8192</td>
<td>2</td>
<td>-</td>
<td>(1:0)</td>
<td>-</td>
<td>(12:0)</td>
<td>16K</td>
</tr>
<tr>
<td>16Kx1</td>
<td>16384</td>
<td>1</td>
<td>-</td>
<td>(0:0)</td>
<td>-</td>
<td>(13:0)</td>
<td>16K</td>
</tr>
</tbody>
</table>
Block RAM Interface

18 Kb Block RAM

Port A
DIA
DIPA
ADDR A
WEA
ENA
RSTA
CLKA
REGCEA

18 Kb Memory Array

Port B
DIB
DIPB
ADDR B
WEB
EN B
RSTB
CLKB
REGCEB

DOA
DOPA
DOB
DOPB
## Block RAM Ports

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI[A</td>
<td>B](1)</td>
</tr>
<tr>
<td>DIP[A</td>
<td>B](1)</td>
</tr>
<tr>
<td>ADDR[A</td>
<td>B]</td>
</tr>
<tr>
<td>WE[A</td>
<td>B]</td>
</tr>
<tr>
<td>EN[A</td>
<td>B]</td>
</tr>
<tr>
<td>RST[A</td>
<td>B]</td>
</tr>
<tr>
<td>CLK[A</td>
<td>B]</td>
</tr>
<tr>
<td>DO[A</td>
<td>B](1)</td>
</tr>
<tr>
<td>DOP[A</td>
<td>B](1)</td>
</tr>
<tr>
<td>REGCE[A</td>
<td>B]</td>
</tr>
</tbody>
</table>
Block RAM Waveforms – READ_FIRST mode

CLK

WE

DI

XXX 1111 2222 XXX

ADDR

aa  bb  cc  dd

DO

0000  MEM(aa) old MEM(bb) old MEM(cc) MEM(dd)

EN

DISABLED READ WRITE MEM(bb)=1111 WRITE MEM(cc)=2222 READ

□S099-2_15_C00403
Block RAM Waveforms – WRITE_FIRST mode

- CLK
- WE
- DI
- ADDR
- DO
- EN

Waveforms:

- XXXX
- 1111
- 2222
- XXXX
- aa
- bb
- cc
- dd
- 0000
- MEM(aa)
- 1111
- MEM(bb)=1111
- 2222
- MEM(cc)=2222
- MEM(dd)

States:

- DISABLED
- READ
- WRITE
  - MEM(bb)=1111
  - MEM(cc)=2222
- READ

DS099-2_14_030403
Block RAM Waveforms – NO_CHANGE mode
## Features of Block RAMs in Spartan-6 FPGAs

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-Port</td>
<td>Yes</td>
</tr>
<tr>
<td>True Dual-Port</td>
<td>Yes</td>
</tr>
<tr>
<td>ROM, Initial RAM Contents</td>
<td>Yes</td>
</tr>
<tr>
<td>Mixed Data Port Widths</td>
<td>Yes</td>
</tr>
<tr>
<td>Power-Up Condition</td>
<td>User-defined data, defaults to zero</td>
</tr>
<tr>
<td>Potential Applications</td>
<td>Local data storage, FIFOs, elastic stores, register files, buffers, stacks, circular buffers, shift registers, delay lines, waveform storage and generation, direct digital synthesis, CAMs, associative memories, function tables, function generators, wide logic functions, code converters, encoders, decoders, counters, state machines, microsequencers, program storage for embedded processor(s)</td>
</tr>
</tbody>
</table>
Inference vs. Instantiation
Inference vs. Instantiation

There are two methods to handle RAMs: instantiation and inference. Many FPGA families provide technology-specific RAMs that you can instantiate in your HDL source code. The software supports instantiation, but you can also set up your source code so that it infers the RAMs. The following table sums up the pros and cons of the two approaches.

<table>
<thead>
<tr>
<th><strong>Inference in Synthesis</strong></th>
<th><strong>Instantiation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advantages</strong></td>
<td><strong>Advantages</strong></td>
</tr>
<tr>
<td>Portable coding style</td>
<td>Most efficient use of the RAM primitives of a specific technology</td>
</tr>
<tr>
<td>Automatic timing-driven synthesis</td>
<td>Supports all kinds of RAMs</td>
</tr>
<tr>
<td>No additional tool dependencies</td>
<td></td>
</tr>
</tbody>
</table>


Using CORE Generator
Generic Inferred ROM
Distributed ROM with asynchronous read

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

Entity ROM is
  generic ( w : integer := 12;
    -- number of bits per ROM word
    r : integer := 3);
    -- 2^r = number of words in ROM
  port (addr : in std_logic_vector(r-1 downto 0);
    dout : out std_logic_vector(w-1 downto 0));
end ROM;
Distributed ROM with asynchronous read

architecture behavioral of rominfr is
    type rom_type is array (0 to 2**r-1)
    of std_logic_vector (w-1 downto 0);
    constant ROM_array : rom_type :=
        ("000011000100",
         "010011010010",
         "010011011011",
         "011011000010",
         "000011110001",
         "011111010110",
         "010011010000",
         "111110011111");
begin
    dout <= ROM_array(to_integer(unsigned(addr)));
end behavioral;
Distributed ROM with asynchronous read

architecture behavioral of rominfr is

  type rom_type is array (0 to 2**r-1)
    of std_logic_vector (w-1 downto 0);

  constant ROM_array : rom_type :=
    (X"0C4",
     X"4D2",
     X"4DB",
     X"6C2",
     X"0F1",
     X"7D6",
     X"4D0",
     X"F9F");

begin
  dout <= ROM_array(to_integer(unsigned(addr)));
end behavioral;
Generic Inferred RAM
Distributed versus Block RAM Inference

Examples:

1. Distributed single-port RAM with asynchronous read

2. Distributed dual-port RAM with asynchronous read

3. Block RAM with synchronous read (no version with asynchronous read!)

More excellent RAM examples from XST Coding Guidelines.
Distributed single-port RAM with asynchronous read

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity raminfr is
  generic ( w : integer := 32;
            -- number of bits per RAM word
            r : integer := 6);
            -- 2^r = number of words in RAM
  port (clk : in std_logic;
        we  : in std_logic;
        a   : in std_logic_vector(r-1 downto 0);
        di  : in std_logic_vector(w-1 downto 0);
        do  : out std_logic_vector(w-1 downto 0));
end raminfr;
Distributed single-port RAM with asynchronous read

architecture behavioral of raminfr is
  type ram_type is array (0 to 2**r-1)
    of std_logic_vector (w-1 downto 0);
  signal RAM : ram_type := (others => (others => '0'));
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if (we = '1') then
        RAM(to_integer(unsigned(a))) <= di;
      end if;
    end if;
  end if;
end process;
  do <= RAM(to_integer(unsigned(a)));
end behavioral;
Distributed dual-port RAM with asynchronous read

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity raminfr is
  generic ( w : integer := 32;
          r : integer := 6);
          -- number of bits per RAM word
          -- 2^r = number of words in RAM

  port (clk  : in std_logic;
       we   : in std_logic;
       a    : in std_logic_vector(r-1 downto 0);
       dpra : in std_logic_vector(r-1 downto 0);
       di   : in std_logic_vector(w-1 downto 0);
       spo  : out std_logic_vector(w-1 downto 0);
       dpo  : out std_logic_vector(w-1 downto 0));

end raminfr;
```
Distributed dual-port RAM with asynchronous read

architecture syn of raminfr is
  type ram_type is array (0 to 2**r-1) of
    std_logic_vector (w-1 downto 0);
  signal RAM : ram_type := (others => (others => '0'));
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if (we = '1') then
        RAM(to_integer(unsigned(a))) <= di;
      end if;
    end if;
  end process;
  spo <= RAM(to_integer(unsigned(a)));
  dpo <= RAM(to_integer(unsigned(dpra)));
end syn;
Block RAM Waveforms – READ_FIRST mode
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity raminfr is
   generic ( w : integer := 32;
             -- number of bits per RAM word
             r : integer := 9);
             -- 2^r = number of words in RAM
   port (clk : in std_logic;
         we  : in std_logic;
         en  : in std_logic;
         addr   : in std_logic_vector(r-1 downto 0);
         di  : in std_logic_vector(w-1 downto 0);
         do  : out std_logic_vector(w-1 downto 0));
end raminfr;
architecture behavioral of raminfr is
    type ram_type is array (0 to 2**r-1) of
      std_logic_vector (w-1 downto 0);
    signal RAM : ram_type := (others => (others => '0'));

begin
    process (clk)
    begin
        if rising_edge(clk) then
            if (en = '1') then
                do <= RAM(to_integer(unsigned(addr)));
                if (we = '1') then
                    RAM(to_integer(unsigned(addr))) <= di;
                end if;
            end if;
        end if;
    end process;
end behavioral;
Block RAM Waveforms – WRITE_FIRST mode

- CLK
- WE
- DI: XXXX 1111 2222 XXXX
- ADDR: aa bb cc dd
- DO: 0000 MEM(aa) 1111 2222 MEM(dd)
- EN: DISABLED READ WRITE MEM(bb)=1111 WRITE MEM(cc)=2222 READ

DS099-2_14_030403
Block RAM with synchronous read
Write-First Mode - cont'd

architecture behavioral of raminfr is
  type ram_type is array (0 to 2**r-1) of
    std_logic_vector (w-1 downto 0);
  signal RAM : ram_type := (others => (others => '0'));

begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (en = '1') then
        if (we = '1') then
          if (we = '1') then
            RAM(to_integer(unsigned(addr))) <= di;
            do <= di;
          else
            do <= RAM(to_integer(unsigned(addr)));
          end if;
        end if;
      end if;
    end if;
  end process;
end behavioral;
Block RAM Waveforms – NO_CHANGE mode

CLK | WE | DI | ADDR | DO | EN

- DISABLED | READ | WRITE MEM(bb)=1111 | WRITE MEM(cc)=2222 | READ

DS099-2_16_030403
Block RAM with synchronous read
No-Change Mode - cont'd

architecture behavioral of raminfr is
  type ram_type is array (0 to 2**r-1) of
    std_logic_vector (w-1 downto 0);
  signal RAM : ram_type := (others => (others => '0'));

begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (en = '1') then
        if (we = '1') then
          RAM(to_integer(unsigned(addr))) <= di;
        else
          do <= RAM(to_integer(unsigned(addr)));
        end if;
      end if;
    end if;
  end process;
end behavioral;
## Criteria for Implementing Inferred RAM in BRAMs

<table>
<thead>
<tr>
<th>Devices</th>
<th>Depth</th>
<th>Depth * Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spartan®-6</td>
<td>&gt;= 127 words</td>
<td>&gt; 512 bits</td>
</tr>
<tr>
<td>Virtex®-6</td>
<td>&gt;= 127 words</td>
<td>&gt; 512 bits</td>
</tr>
<tr>
<td>7 series</td>
<td>&gt;= 127 words</td>
<td>&gt; 512 bits</td>
</tr>
</tbody>
</table>