FPGA Embedded Resources

Recommended reading

- XAPP463 Using Block RAM in Spartan-3 Generation FPGAs
  Google search: XAPP463
- XAPP464 Using Look-Up Tables as Distributed RAM in Spartan-3 Generation FPGAs
  Google search: XAPP464
- XST User Guide, Section: Coding Techniques
  Google search: XST User Guide (PDF)
  http://www.xilinx.com/ipcenter/data/docs/xst/hdlcode.html (HTML)
- ISE In-Depth Tutorial, Section: Creating a CORE Generator Module
  Google search: ISE In-Depth Tutorial

Xilinx FPGA Devices

<table>
<thead>
<tr>
<th>Technology</th>
<th>Low-cost</th>
<th>High-performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>120/150 nm</td>
<td></td>
<td>Virtex 2, 2 Pro</td>
</tr>
<tr>
<td>90 nm</td>
<td>Spartan 3</td>
<td>Virtex 4</td>
</tr>
<tr>
<td>65 nm</td>
<td></td>
<td>Virtex 5</td>
</tr>
<tr>
<td>45 nm</td>
<td>Spartan 6</td>
<td></td>
</tr>
<tr>
<td>40 nm</td>
<td></td>
<td>Virtex 6</td>
</tr>
</tbody>
</table>

Altera FPGA Devices

<table>
<thead>
<tr>
<th>Technology</th>
<th>Low-cost</th>
<th>Mid-range</th>
<th>High-performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>130 nm</td>
<td>Cyclone</td>
<td></td>
<td>Stratix</td>
</tr>
<tr>
<td>90 nm</td>
<td>Cyclone II</td>
<td></td>
<td>Stratix II</td>
</tr>
<tr>
<td>65 nm</td>
<td>Cyclone III</td>
<td></td>
<td>Arria I</td>
</tr>
<tr>
<td>40 nm</td>
<td>Cyclone IV</td>
<td>Arria II</td>
<td>Stratix IV</td>
</tr>
</tbody>
</table>
Number of Multipliers per Spartan 3 Device

<table>
<thead>
<tr>
<th>Device</th>
<th>Multiplier Columns</th>
<th>Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC3S50</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>XC3S200</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>XC3S400</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>XC3S1000</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>XC3S1500</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>XC3S2000</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>XC3S4000</td>
<td>4</td>
<td>96</td>
</tr>
<tr>
<td>XC3S5000</td>
<td>4</td>
<td>104</td>
</tr>
</tbody>
</table>

Combinational and Registered Multiplier
### Dedicated Multiplier Block

![Diagram of Dedicated Multiplier Block](image1.png)

### Interface of a Dedicated Multiplier

![Diagram of Interface of a Dedicated Multiplier](image2.png)

### Unsigned vs. Signed Multiplication

<table>
<thead>
<tr>
<th></th>
<th>Unsigned</th>
<th>Signed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>15</td>
<td>1111</td>
</tr>
<tr>
<td>x 1111</td>
<td>x 15</td>
<td>x -1</td>
</tr>
</tbody>
</table>
| 11100001| 225   | 00000001| 1

### Embedded Multiplier Block Overview

Each Cyclone II has one to three columns of embedded multipliers.

Each embedded multiplier can be configured to support:
- One 18 x 18 multiplier
- Two 9 x 9 multipliers

![Diagram of Embedded Multiplier Block](image3.png)

### Number of Embedded Multipliers

<table>
<thead>
<tr>
<th>Device</th>
<th>Embedded Multipliers</th>
<th>0 × 9 Multipliers</th>
<th>18 × 18 Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP2C5</td>
<td>13</td>
<td>26</td>
<td>13</td>
</tr>
<tr>
<td>EP2C8</td>
<td>18</td>
<td>36</td>
<td>18</td>
</tr>
<tr>
<td>EP2C20</td>
<td>26</td>
<td>52</td>
<td>26</td>
</tr>
<tr>
<td>EP2C35</td>
<td>36</td>
<td>70</td>
<td>35</td>
</tr>
<tr>
<td>EP2C50</td>
<td>86</td>
<td>172</td>
<td>86</td>
</tr>
<tr>
<td>EP2C70</td>
<td>150</td>
<td>300</td>
<td>150</td>
</tr>
</tbody>
</table>

![Table of Embedded Multipliers](image4.png)
Multiplier Block Architecture

Two Multiplier Types

Multiplier Stage

• Signals $\text{signa}$ and $\text{signb}$ are used to identify the signed and unsigned inputs.

<table>
<thead>
<tr>
<th>Multiplier Sign Representation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Data A</td>
<td>Data B</td>
</tr>
<tr>
<td>$\text{signa}$</td>
<td>$\text{Logic Level}$</td>
</tr>
<tr>
<td>Signed</td>
<td>Low</td>
</tr>
<tr>
<td>Signed</td>
<td>High</td>
</tr>
<tr>
<td>Signed</td>
<td>High</td>
</tr>
</tbody>
</table>

3 Ways to Use Dedicated Hardware

• Three (3) ways to use dedicated (embedded) hardware
  – Inference
  – Instantiation
  – CoreGen in Xilinx
  MegaWizard Plug-In Manager in Altera

Inferred Multiplier

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

entity mult18x18 is
  generic (word_size : natural   := 18;
            signed_mult : boolean := true);
  port (clk : in std_logic;
         a   : in    std_logic_vector(word_size-1 downto 0);
         b   : in    std_logic_vector(word_size-1 downto 0);
         c   : out   std_logic_vector(2*word_size-1 downto 0));
end entity mult18x18;

architecture infer of mult18x18 is
begin
  process (clk)
  begin
    if rising_edge (clk) then
      if signed_mult then
        c <= std_logic_vector(signed(a) * signed(b));
      else
        c <= std_logic_vector(unsigned(a) * unsigned(b));
      end if;
    end if;
  end process;
end architecture infer;
```

Forcing a particular implementation in VHDL

Synthesis tool: Xilinx XST

```
Attribute MULT_STYLE: string;
Attribute MULT_STYLE of c: signal is block;
```

Allowed values of the attribute:
block – dedicated multiplier
lut - LUT-based multiplier
pipe_block – pipelined dedicated multiplier
pipe_lut – pipelined LUT-based multiplier
auto – automatic choice by the synthesis tool
Instantiation for Spartan 3 FPGAs

VHDL Instantiation Template
-- Component Declaration for MULTI8X18 should be placed
-- after architecture statement but before begin keyword
component MULTI8X18
  port ( P : out STD_LOGIC_VECTOR (35 downto 0));
  A : in STD_LOGIC_VECTOR (17 downto 0);
  B : in STD_LOGIC_VECTOR (17 downto 0));
end component;

-- Component Instantiation for MULTI8X18 should be placed
-- in architecture after the begin keyword
MULTI8X18 instantiate MULTI8X18
port map (P => user_P,
  A => user_A,
  B => user_B);

CORE Generator

Xilinx XtremeDSP
• Starting with Virtex 4 family, Xilinx introduced DSP48 block for high-speed DSP on FPGAs
• Essentially a multiply-accumulate core with many other features
• Now also in Spartan-3A, Spartan 6, Virtex 5, and Virtex 6

DSP48 Slice: Virtex 4

Simplified Form of DSP48

$$Adder\ Out = (X + (Y + CIN))$$
Choosing Inputs to DSP Adder

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>0</td>
<td>000</td>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>PCin</td>
<td>001</td>
<td>AxB</td>
<td>01</td>
<td>AxB</td>
<td>01</td>
</tr>
<tr>
<td>P</td>
<td>010</td>
<td>InvSum</td>
<td>10</td>
<td>P</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>011</td>
<td>C</td>
<td>11</td>
<td>AxB</td>
<td>11</td>
</tr>
<tr>
<td>Shift/PCin</td>
<td>101</td>
<td>Shift</td>
<td>110</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ P = \text{Adder Out} = (Z \pm (X + Y + CIN)) \]

DSP48E Slice : Virtex5

New in Virtex 5 Compared to Virtex 4

Memory Types

- Memory
  - RAM
  - ROM
- Memory
  - Single port
  - Dual port
- Memory
  - With asynchronous read
  - With synchronous read

Embedded Memories
### Memory Types in Xilinx

- **Distributed (MLUT-based)**
- **Block RAM-based (BRAM-based)**
- **Inferred**
- **Instantiated**
- **Manually Using Core Generator**

### Memory Types in Altera

- **Distributed (ALUT-based, Stratix III onwards)**
- **Memory block-based**
- **Small size** (512)
- **Medium size** (4K, 9K, 20K)
- **Large size** (144K, 512K)
- **Inferred**
- **Instantiated**
- **Manually Using MegaWizard Plug-In Manager**

### 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>Inference in Synthesis</th>
<th>Instantiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantages</td>
<td>Advantages</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>

### FPGA Distributed Memory

### CLB Slice
Distributed RAM

- CLB LUT configurable as Distributed RAM
  - An LUT equals 16x1 RAM
  - Cascade LUTs to increase RAM size
- Synchronous write
- Asynchronous read
  - Can create a synchronous read by using extra flip-flops
  - Naturally, distributed RAM read is asynchronous
- Two LUTs can make
  - 32 x 1 single-port RAM
  - 16 x 2 single-port RAM
  - 16 x 1 dual-port RAM

Block RAM

- Most efficient memory implementation
- Dedicated blocks of memory
- Ideal for most memory requirements
  - 4 to 104 memory blocks
  - 18 kbits = 18,432 bits per block (16 k without parity bits)
  - Use multiple blocks for larger memories
- Builds both single and true dual-port RAMs
- Synchronous write and read (different from distributed RAM)

Spartan-3E Block RAM Amounts

<table>
<thead>
<tr>
<th>Device</th>
<th>Total Number of RAM Blocks</th>
<th>Total Addressable Locations (bits)</th>
<th>Number of Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC3S100E</td>
<td>4</td>
<td>73,728</td>
<td>1</td>
</tr>
<tr>
<td>XC3S250E</td>
<td>12</td>
<td>221,184</td>
<td>2</td>
</tr>
<tr>
<td>XC3S500E</td>
<td>20</td>
<td>368,640</td>
<td>2</td>
</tr>
<tr>
<td>XC3S1200E</td>
<td>28</td>
<td>516,096</td>
<td>2</td>
</tr>
<tr>
<td>XC3S1600E</td>
<td>36</td>
<td>663,552</td>
<td>2</td>
</tr>
</tbody>
</table>
Block RAM can have various configurations (port aspect ratios)

<table>
<thead>
<tr>
<th>Port A In</th>
<th>Port A Out</th>
<th>Port B In</th>
<th>Port B Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K-Bit Depth</td>
<td>18-Bit Width</td>
<td>8K-Bit Depth</td>
<td>9-Bit Width</td>
</tr>
<tr>
<td>Port A In</td>
<td>Port A Out</td>
<td>Port B In</td>
<td>Port B Out</td>
</tr>
<tr>
<td>WEA</td>
<td>ENA</td>
<td>RSTA</td>
<td>ADDRA[12:0]</td>
</tr>
</tbody>
</table>

Single-Port Block RAM

<table>
<thead>
<tr>
<th>Port A In</th>
<th>Port A Out</th>
<th>Port B In</th>
<th>Port B Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K-Bit Depth</td>
<td>18-Bit Width</td>
<td>8K-Bit Depth</td>
<td>9-Bit Width</td>
</tr>
<tr>
<td>Port A In</td>
<td>Port A Out</td>
<td>Port B In</td>
<td>Port B Out</td>
</tr>
<tr>
<td>WEA</td>
<td>ENA</td>
<td>RSTA</td>
<td>ADDRA[12:0]</td>
</tr>
</tbody>
</table>

Dual-Port Bus Flexibility

- Each port can be configured with a different data bus width
- Provides easy data width conversion without any additional logic

Two Independent Single-Port RAMs

- Added advantage of True Dual-Port
  - No wasted RAM bits
  - Can split a Dual-Port 16K RAM into two Single-Port 8K RAM
- Simultaneous independent access to each RAM

- To access the lower RAM
  - Tie the MSB address bit to Logic Low
- To access the upper RAM
  - Tie the MSB address bit to Logic High
The embedded memory structure consists of columns of M4K memory blocks that can be configured as RAM, first-in first-out (FIFO) buffers, and ROM.

### Memory Modes

The M4K memory blocks support the following modes:

- Single-port RAM (RAM:1-Port)
- Simple dual-port RAM (RAM: 2-Port)
- True dual-port RAM (RAM:2-Port)
- Tri-port RAM (RAM:3-Port)
- Single-port ROM (ROM:1-Port)
- Dual-port ROM (ROM:2-Port)

### Single-Port ROM

- The address lines of the ROM are registered
- The outputs can be registered or unregistered
- A `.mif` file is used to initialize the ROM contents

### Stratix II TriMatrix Memory

<table>
<thead>
<tr>
<th>Feature</th>
<th>M512 Blocks</th>
<th>M4K Blocks</th>
<th>M16K Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum performance</td>
<td>600 MHz</td>
<td>550 MHz</td>
<td>400 MHz</td>
</tr>
<tr>
<td>Total RAM bits (including parity bits)</td>
<td>512</td>
<td>4,096</td>
<td>5,091,024</td>
</tr>
<tr>
<td>Configurations</td>
<td>512 x 1</td>
<td>256 x 2</td>
<td>64 x 9</td>
</tr>
<tr>
<td>Total RAM bits (including parity bits)</td>
<td>128 x 4</td>
<td>64 x 9</td>
<td>32 x 18</td>
</tr>
<tr>
<td>Configurations</td>
<td>256 x 16</td>
<td>128 x 32</td>
<td>64 x 18</td>
</tr>
<tr>
<td>Total RAM bits (including parity bits)</td>
<td>256 x 16</td>
<td>128 x 32</td>
<td>64 x 18</td>
</tr>
</tbody>
</table>

### Stratix III & Stratix IV TriMatrix Memory

<table>
<thead>
<tr>
<th>Feature</th>
<th>M64K</th>
<th>M128K</th>
<th>M64K Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum performance</td>
<td>600 MHz</td>
<td>500 MHz</td>
<td>380 MHz</td>
</tr>
<tr>
<td>Total memory bits (including parity bits)</td>
<td>540 (or ROM only) or 256 (or other modes)</td>
<td>512</td>
<td>147,636</td>
</tr>
<tr>
<td>Configurations (depth = width)</td>
<td>16 x 16</td>
<td>8 x 8</td>
<td>8 x 8</td>
</tr>
<tr>
<td>Total memory bits (including parity bits)</td>
<td>16 x 16</td>
<td>8 x 8</td>
<td>8 x 8</td>
</tr>
<tr>
<td>Configurations</td>
<td>16 x 16</td>
<td>8 x 8</td>
<td>8 x 8</td>
</tr>
<tr>
<td>Total memory bits (including parity bits)</td>
<td>16 x 16</td>
<td>8 x 8</td>
<td>8 x 8</td>
</tr>
</tbody>
</table>
Generic Multiplier (1)

entity mult is
  generic
    vendor : integer := XILINX;  -- vendor : XILINX=0, ALTERA=1
    multiplier_type : integer := MUL_DEDICATED;  -- multiplier_type : MUL_LOGIC_BASED=0, MUL_DSP_BASED=1
    WIDTH : integer := 8;  -- width : width (fixed width for input and output)
  port
    a : in std_logic_vector (WIDTH-1 downto 0);
    b : in std_logic_vector (WIDTH-1 downto 0);
    s : out std_logic_vector (WIDTH-1 downto 0);
  end mult;

architecture mult of mult is
begin
  if (multiplier_type = MUL_DEDICATED and vendor = XILINX) generate
    vendor_mult_gen : entity work.mult(a, b, s) generic map (WIDTH => WIDTH)
    port map (a => a, b => b, s => s);
  end generate;
  if (multiplier_type = MUL_LOGIC_BASED and vendor = XILINX) generate
    vendor_mult_gen : entity work.mult(a, b, s) generic map (WIDTH => WIDTH)
    port map (a => a, b => b, s => s);
  end generate;
end generate;
end mult;
**Generic Multiplier (3)**

```vhdl
architecture xilinx_logic of mult is
signal temp1 : std_logic_vector(2*WIDTH-1 downto 0);
attribute mult_style : string;
attribute mult_style of temp1 : signal is "lut";
begins
  temp1 <= STD_LOGIC_VECTOR(unsigned(a) * unsigned(b));
  s <= temp1(WIDTH-1 downto 0);
end xilinx_logic;
```

**Generic Multiplier (4)**

```vhdl
architecture altera_logic of mult is
signal temp : std_logic_vector(2*WIDTH-1 downto 0);
attribute multstyle : string;
attribute multstyle of altera_logic : architecture is "logic";
begins
  temp <= STD_LOGIC_VECTOR(unsigned(a) * unsigned(b));
  s <= temp(WIDTH-1 downto 0);
end altera_logic;
```

**FPGA Distributed Memory**

- CLB LUT configurable as Distributed RAM
- An LUT equals 16x1 RAM
- Cascade LUTs to increase RAM size
- Synchronous write
- Asynchronous read
  - Can create a synchronous read by using extra flip-flops
  - Naturally, distributed RAM read is asynchronous
- Two LUTs can make
  - 32 x 1 single-port RAM
  - 16 x 2 single-port RAM
  - 16 x 1 dual-port RAM
### 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>Advantages</th>
<th>Inference in Synthesis</th>
<th>Instantiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portable coding style</td>
<td></td>
<td>Most efficient use of the RAM primitives</td>
</tr>
<tr>
<td>Automatic timing-driven</td>
<td></td>
<td>of a specific technology</td>
</tr>
<tr>
<td>synthesis</td>
<td></td>
<td>Supports all kinds of RAMs</td>
</tr>
<tr>
<td>No additional tool dependencies</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Distributed versus Block RAM Inference

**Examples:**

1. Distributed single-port RAM with asynchronous read
2. Distributed dual-port RAM with asynchronous read
3. Distributed single-port RAM with "false" synchronous read
4. Block RAM with synchronous read (no version with asynchronous read!)

More excellent RAM examples from XST Coding Guidelines:

(Click on RAMs)

#### Distributed RAM with asynchronous read

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
entity raminfr is
  generic ( data_bits : integer := 32 ;
            -- number of bits per RAM word
            addr_bits : integer := 3 ;
            -- 2^addr_bits = number of words in RAM
            );
  port ( clk : in std_logic ;
         we : in std_logic ;
         a   : in std_logic_vector(addr_bits-1 downto 0);
         di  : in std_logic_vector(data_bits-1 downto 0);
         do  : out std_logic_vector(data_bits-1 downto 0));
end raminfr;
```

#### Distributed single-port RAM with asynchronous read

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
entity raminfr is
  generic ( data_bits : integer := 32 ;
            -- number of bits per RAM word
            addr_bits : integer := 3 ;
            -- 2^addr_bits = number of words in RAM
            );
  port ( clk : in std_logic ;
         we : in std_logic ;
         a   : in std_logic_vector(addr_bits-1 downto 0);
         di  : in std_logic_vector(data_bits-1 downto 0);
         do  : out std_logic_vector(data_bits-1 downto 0));
end raminfr;
```
Distributed single-port RAM with asynchronous read

architecture behavioral of raminfr is
  type ram_type is array (2**addr_bits-1 downto 0) of std_logic_vector (data_bits-1 downto 0);
n signal RAM : ram_type;
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (we = '1') then
        RAM(conv_integer(unsigned(a))) <= di;
      end if;
    end if;
  end process;
  end behavioral;

Report from Synthesis

Resource Usage Report for raminfr
Mapping to part: xc3s50pq208-5
Cell usage:
  GND             1 use
  RAM16X4S        8 uses
  I/O ports: 69
  I/O primitives: 68
  IBUF           36 uses
  OBUF           32 uses
  BUFGP          1 use
  IC Register bits: 0
  Register bit not including I/Os: 0 (0%)

RAM/ROM usage summary
  Single Port Rams (RAM16X4S): 8
  Global Clock Buffers: 1 of 8 (12%)

Mapping Summary:
  Total LUTs: 32 (2%)

Distributed dual-port RAM with asynchronous read

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity raminfr is
  generic ( data_bits : integer := 32;
              -- number of bits per RAM word
              addr_bits : integer := 3);
  port (clk  : in std_logic;
         we   : in std_logic;
         a    : in std_logic_vector(addr_bits-1 downto 0);
         dpra : in std_logic_vector(addr_bits-1 downto 0);
         di   : in std_logic_vector(data_bits-1 downto 0);
         spo  : out std_logic_vector(data_bits-1 downto 0);
         dpo  : out std_logic_vector(data_bits-1 downto 0));
end raminfr;

architecture syn of raminfr is
  type ram_type is array (2**addr_bits-1 downto 0) of std_logic_vector (data_bits-1 downto 0);
  signal RAM : ram_type;
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (we = '1') then
        RAM(conv_integer(unsigned(a))) <= di;
      end if;
    end if;
  end process;
  end behavioral;
end syn;

Report from Implementation

Design Summary:
  Number of errors: 0
  Number of warnings: 0
  Logic utilization:
    Logic distribution:
      Number of occupied Slices: 16 out of 768 2%
      Number of Slices containing only related logic: 16 out of 16 100%
      Number of Slices containing unrelated logic: 0 out of 16 0%
  *See NOTES below for an explanation of the effects of unrelated logic
  Total Number of 4 input LUTs: 32 out of 1,536 2%
  Number used as 16x1 RAM: 32
  Number of bonded I/Os: 69 out of 124 55%
  Number of GCLKs: 1 out of 8 12%

Distributed dual-port RAM with asynchronous read

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity raminfr is
  generic ( data_bits : integer := 32;
              -- number of bits per RAM word
              addr_bits : integer := 3);
  port (clk  : in std_logic;
         we   : in std_logic;
         a    : in std_logic_vector(addr_bits-1 downto 0);
         dpra : in std_logic_vector(addr_bits-1 downto 0);
         di   : in std_logic_vector(data_bits-1 downto 0);
         spo  : out std_logic_vector(data_bits-1 downto 0);
         dpo  : out std_logic_vector(data_bits-1 downto 0));
end raminfr;

architecture syn of raminfr is
  type ram_type is array (2**addr_bits-1 downto 0) of std_logic_vector (data_bits-1 downto 0);
  signal RAM : ram_type;
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (we = '1') then
        RAM(conv_integer(unsigned(a))) <= di;
      end if;
    end if;
  end process;
  end behavioral;
end syn;
Report from Synthesis

Resource Usage Report for raminfr
Mapping to part: xc3s50pq208-5
Cell usage:

- GND: 1 use
- I/O pins: 103
- IBUF: 36 uses
- OBUF: 64 uses
- BUFGP: 1 use
- I/O Register bits: 0 use
- RAM/ROM usage summary:
  - Dual Port Rams (RAM16X1D): 32
  - Global Clock Buffers: 1 of 8 (12%)
Mapping Summary:
- Total LUTs: 4% (32)

Report from Implementation

Design Summary:
- Number of errors: 0
- Number of warnings: 0
Logic Utilization:
- Logic Distribution:
  - Number of occupied cells: 32 out of 768 (4%)
  - Number of Slices containing only related logic: 32 out of 32 (100%)
  - Number of Slices containing unrelated logic: 0 out of 32 (0%)
  - See NOTES below for an explanation of the effects of unrelated logic
  - Total Number of 4 input LUTs: 64 out of 1,536 (4%)
  - Two LUTs used per Dual Port RAM
- Number of bonded IOBs: 104 out of 124 (83%)
- Number of GCLKs: 1 out of 8 (12%)

Distributed RAM with "false" synchronous read

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
entity raminfr is
  generic (data_bits : integer := 32;
            addr_bits : integer := 3);
  port (clk : in std_logic;
        we : in std_logic;
        a   : in std_logic_vector(addr_bits-1 downto 0);
        di  : in std_logic_vector(data_bits-1 downto 0);
        do  : out std_logic_vector(data_bits-1 downto 0));
end raminfr;

architecture behavioral of raminfr is
  type ram_type is array (2**addr_bits-1 downto 0)
    of std_logic_vector (data_bits-1 downto 0);
  signal RAM : ram_type;
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (we = '1') then
        RAM(conv_integer(unsigned(a))) <= di;
      end if;
      do <= RAM(conv_integer(unsigned(a)));
    end if;
  end process;
end behavioral;
**Report from Implementation**

**Design Summary:**
- **Number of errors:** 0
- **Number of warnings:** 0

**Logic Utilization:**
- **Number of Slice Flip Flops:** 32 out of 1,536 (2%)
- **Logic Distribution:**
  - **Number of occupied Slices:** 16 out of 768 (2%)
  - **Number of Slices containing only related logic:** 16 out of 16 (100%)
  - **Number of Slices containing unrelated logic:** 0 out of 16 (0%)
  - *See NOTES below for an explanation of the effects of unrelated logic*

**Total Number of 4 input LUTs:** 32 out of 1,536 (2%)
- **Number used as 16x1 RAMs:** 32
- **Number of bonded IOBs:** 69 out of 124 (55%)
- **Number of GCLKs:** 1 out of 8 (12%)

**Total equivalent gate count for design:** 4,355

---

**Block RAM with synchronous read**

*(write first mode)*

**LIBRARY ieee;**
**USE ieee.std_logic_1164.all;**
**USE ieee.std_logic_arith.all;**

**entity raminfr is**
**generic (data_bits : integer := 32;**
- **number of bits per RAM word**
- **addr_bits : integer := 3;**
- 2^addr_bits = number of words in RAM
**port (clk : in std_logic;**
- **we : in std_logic;**
- **a : in std_logic_vector(addr_bits-1 downto 0);**
- **di : in std_logic_vector(data_bits-1 downto 0);**
- **do : out std_logic_vector(data_bits-1 downto 0));**
**end raminfr;**

**architecture behavioral of raminfr is**
**type ram_type is array (2**^addr_bits-1 downto 0) of std_logic_vector(data_bits-1 downto 0);**
**signal RAM : ram_type;**
**signal read_a : std_logic_vector(addr_bits-1 downto 0);**

**begin**
**process (clk)**
**begin**
**if (clk'event and clk = '1') then**
**if (we = '1') then**
**RAM(conv_integer(unsigned(a))) <= di;**
**end if;**
**read_a <= a;**
**end if;**
**end process;**
**do <= RAM(conv_integer(unsigned(read_a)));**
**end behavioral;**

---

**Block RAM Waveforms – WRITE_FIRST mode**

---

**Block RAM with synchronous read**

*(write first mode)*

**architecture behavioral of raminfr is**
**type ram_type is array (2^addr_bits-1 downto 0) of std_logic_vector(data_bits-1 downto 0);**
**signal RAM : ram_type;**
**signal read_a : std_logic_vector(addr_bits-1 downto 0);**

**begin**
**process (clk)**
**begin**
**if (clk'event and clk = '1') then**
**if (we = '1') then**
**RAM(conv_integer(unsigned(a))) <= di;**
**end if;**
**end if;**
**end process;**
**do <= RAM(conv_integer(unsigned(read_a)));**
**end behavioral;**

---

**Report from Synthesis**

**Resource Usage Report for raminfr**
Mapping to part: xc3s50pq208-5
**Cell usage:**
- **GND:** 1 use
- **RAMB16_S36:** 1 use
- **VCC:** 1 use
- **IBUF:** 36 uses
- **OBUF:** 32 uses
- **BUFGP:** 1 use
- **I/O Register bits:** 0
- **Register bits not including I/Os:** 0 (0%)

**RAM/ROM usage summary:**
- **Block RAM:** 1 of 1 (100%)
- **Global Clock Buffers:** 1 of 8 (12%)

**Mapping Summary:**
- **Total LUTs:** 0 (0%)
Report from Implementation

Design Summary:
Number of errors: 0
Number of warnings: 0
Logic Utilization:
- Number of Slices containing only related logic: 0 out of 0 0%
- Number of Slices containing unrelated logic: 0 out of 0 0%
*See NOTES below for an explanation of the effects of unrelated logic
- Number of bonded IOBs: 69 out of 124 55%
- Number of Block RAMs: 1 out of 4 25%
- Number of GCLKs: 1 out of 8 12%

Distributed ROM with asynchronous read

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

entity rominfr is
  generic (data_bits : integer := 10; -- number of bits per ROM word
            addr_bits : integer := 3); -- 2^addr_bits = number of words in ROM
  port (a : in std_logic_vector(addr_bits-1 downto 0);
        do : out std_logic_vector(data_bits-1 downto 0));
end rominfr;

architecture behavioral of rominfr is
  type rom_type is array (2**addr_bits-1 downto 0) of std_logic_vector (data_bits-1 downto 0);
  constant ROM : rom_type :=
  (
    "0000110001",
    "0100110100",
    "0100110110",
    "0110110000",
    "0000111100",
    "0111110101",
    "0100110100",
    "1111100111"
  );
begin
  do <= ROM(conv_integer(unsigned(a)));
end behavioral;

Distributed ROM with asynchronous read

architecture behavioral of rominfr is
  type rom_type is array (2**addr_bits-1 downto 0) of std_logic_vector (data_bits-1 downto 0);
  constant ROM : rom_type :=
  (
    "0000110001",
    "0100110100",
    "0100110110",
    "0110110000",
    "0000111100",
    "0111110101",
    "0100110100",
    "1111100111"
  );
begin
  do <= ROM(conv_integer(unsigned(a)));
end behavioral;

Using CORE Generator
library IEEE;
use IEEE.STD_LOGIC_1164.all;

library UNISIM;
use UNISIM.all;

entity RAM_16x1_DISTRIBUTED is
port(
    CLK : in STD_LOGIC;
    WE : in STD_LOGIC;
    ADDR : in STD_LOGIC_VECTOR(3 downto 0);
    DATA_IN : in STD_LOGIC;
    DATA_OUT : out STD_LOGIC
);
end RAM_16x1_DISTRIBUTED;

architecture RAM_16x1_DISTRIBUTED_STRUCTURAL of RAM_16x1_DISTRIBUTED is
-- part used by the synthesis tool, Synplify Pro, only;
-- ignored during simulation
attribute INIT : string;
attribute INIT of RAM_16x1s : label is "0000";
component ram16x1s
    generic(
        INIT : BIT_VECTOR(15 downto 0) := X"0000"
    );
    port(
        O : out std_ulogic;
        A0 : in std_ulogic;
        A1 : in std_ulogic;
        A2 : in std_ulogic;
        A3 : in std_ulogic;
        D : in std_ulogic;
        WCLK : in std_ulogic;
        WE : in std_ulogic
    );
end component;
begin
    RAM_16x1s_1 : ram16x1s generic map (INIT => X"0000")
port map(
    O => DATA_OUT,
    A0 => ADDR(0),
    A1 => ADDR(1),
    A2 => ADDR(2),
    A3 => ADDR(3),
    D => DATA_IN,
    WCLK => CLK,
    WE => WE
);
end RAM_16x1_DISTRIBUTED_STRUCTURAL;

begin
    RAM_16x8s_1 : ram16x8s generic map (INIT => X"0000"
port map(
    O => DATA_OUT,
    ADDR : in STD_LOGIC_VECTOR(3 downto 0);
    DATA_IN : in STD_LOGIC_VECTOR(7 downto 0);
    DATA_OUT : out STD_LOGIC_VECTOR(7 downto 0)
);
end RAM_16x8_DISTRIBUTED:

begin
    RAM_16x8s_1 : ram16x8s generic map (INIT => X"0000"
port map(
    O => DATA_OUT,
    ADDR : in STD_LOGIC_VECTOR(3 downto 0);
    DATA_IN : in STD_LOGIC_VECTOR(7 downto 0);
    DATA_OUT : out STD_LOGIC_VECTOR(7 downto 0)
);
end RAM_16x8_DISTRIBUTED;
### RAM 16x8 (2)

```vhdl
architecture RAM_16X_DISTRIBUTED_STRUCTURAL of RAM_16X_DISTRIBUTED is
  -- part used by the synthesis tool, Synplify Pro, only;
  -- ignored during simulation
  attribute INIT : string;
  attribute INIT of RAM_16X : label is "0000";
  component ram_16x8
    generic:
      INIT : BIT_VECTOR (15 downto 0) := X"0000";
    port:
      O : out std_logic;
      A0 : in std_logic;
      A1 : in std_logic;
      A2 : in std_logic;
      A3 : in std_logic;
      D : in std_logic;
      WCLK : in std_logic;
      WE : in std_logic;
    end component;
begin
  GENERATE_MEMORY:
    for I in 0 to 7 generate
      RAM_16X_S_1:
        ram_16x8
          generic map (INIT => X"0000")
          port map
            (O => DATA_OUT(I),
             A0 => ADDR(0),
             A1 => ADDR(1),
             A2 => ADDR(2),
             A3 => ADDR(3),
             D => DATA_IN(I),
             WCLK => CLK,
             WE => WE);
    end generate;
end RAM_16X8_DISTRIBUTED_STRUCTURAL;
```

### RAM 16x8 (3)

```vhdl
begin
  GENERATE_MEMORY:
    for I in 0 to 7 generate
      RAM_16X_S_1: ram_16x8
        generic map (INIT => X"0000")
        port map
          (O => DATA_OUT(I),
           A0 => ADDR(0),
           A1 => ADDR(1),
           A2 => ADDR(2),
           A3 => ADDR(3),
           D => DATA_IN(I),
           WCLK => CLK,
           WE => WE);
    end generate;
end RAM_16X8_DISTRIBUTED_STRUCTURAL;
```

### ROM 16x1 (1)

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
library UNISIM;
use UNISIM.all;
entity ROM_16X1_DISTRIBUTED is
  port(
    ADDR : in STD_LOGIC_VECTOR (3 downto 0);
    DATA_OUT : out STD_LOGIC
  );
end ROM_16X1_DISTRIBUTED;
architecture ROM_16X1_DISTRIBUTED_STRUCTURAL of ROM_16X1_DISTRIBUTED is
  -- part used by the synthesis tool, Synplify Pro, only;
  -- ignored during simulation
  attribute INIT : string;
  attribute INIT of rom_16x1s_1: label is "F0C1";
  component rom_16x1s
    generic:
      INIT : BIT_VECTOR (15 downto 0) := X"0000";
    port:
      O : out std_logic;
      A0 : in std_logic;
      A1 : in std_logic;
      A2 : in std_logic;
      D : in std_logic;
      WCLK : in std_logic;
      WE : in std_logic;
    end component;
    signal Low : std_logic := '0';
begin
  rom_16x1s_1: rom_16x1s
    generic map (
      INIT => X"F0C1"
    )
    port map
      (O => DATA_OUT, A0 => ADDR(0), A1 => ADDR(1), A2 => ADDR(2), A3 => ADDR(3), D => Low, WCLK => Low, WE => Low);
end ROM_16X1_DISTRIBUTED_STRUCTURAL;
```

### ROM 16x1 (2)

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
library UNISIM;
use UNISIM.all;
entity ROM_16X1_DISTRIBUTED is
  port(
    ADDR : in STD_LOGIC_VECTOR (3 downto 0);
    DATA_OUT : out STD_LOGIC
  );
end ROM_16X1_DISTRIBUTED;
architecture ROM_16X1_DISTRIBUTED_STRUCTURAL of ROM_16X1_DISTRIBUTED is
  -- part used by the synthesis tool, Synplify Pro, only;
  -- ignored during simulation
  attribute INIT : string;
  attribute INIT of rom_16x1s_1: label is "F0C1";
  component rom_16x1s
    generic:
      INIT : BIT_VECTOR (15 downto 0) := X"0000";
    port:
      O : out std_logic;
      A0 : in std_logic;
      A1 : in std_logic;
      A2 : in std_logic;
      D : in std_logic;
      WCLK : in std_logic;
      WE : in std_logic;
    end component;
    signal Low : std_logic := '0';
begin
  ROM_16X1_DISTRIBUTED_STRUCTURAL:
    rom_16x1s_1: rom_16x1s
      generic map (
        INIT => X"F0C1"
      )
      port map
        (O => DATA_OUT, A0 => ADDR(0), A1 => ADDR(1), A2 => ADDR(2), A3 => ADDR(3), D => Low, WCLK => Low, WE => Low);
end ROM_16X1_DISTRIBUTED_STRUCTURAL;
```

### Block RAM library components

<table>
<thead>
<tr>
<th>Component</th>
<th>Data Cells</th>
<th>Parity Cells</th>
<th>Address Bus</th>
<th>Data Bus</th>
<th>Parity Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Depth</td>
<td>Width</td>
<td>Depth</td>
<td>Width</td>
<td></td>
</tr>
<tr>
<td>RAM16_64</td>
<td>4096</td>
<td>256</td>
<td>(13:0)</td>
<td>(0:0)</td>
<td></td>
</tr>
<tr>
<td>RAM16_32</td>
<td>2048</td>
<td>128</td>
<td>(12:0)</td>
<td>(0:0)</td>
<td></td>
</tr>
<tr>
<td>RAM16_16</td>
<td>1024</td>
<td>64</td>
<td>(11:0)</td>
<td>(0:0)</td>
<td></td>
</tr>
<tr>
<td>RAM16_8</td>
<td>512</td>
<td>32</td>
<td>(10:0)</td>
<td>(0:0)</td>
<td>(0:0)</td>
</tr>
<tr>
<td>RAM16_4</td>
<td>256</td>
<td>16</td>
<td>(9:0)</td>
<td>(0:0)</td>
<td>(0:0)</td>
</tr>
</tbody>
</table>
Component declaration for BRAM (1)

```
component RAMB16_S1
    -- synthesis translate_off
    generic (
        INIT : bit_vector := X"0";
        INIT_00 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INIT_02 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INIT_03 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INIT_04 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INIT_05 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INIT_06 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INIT_07 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INIT_1F : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INITP_00 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INITP_07 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        SRVAL : bit_vector := X"0";
        WRITE_MODE : string := "WRITE_FIRST";
    );
    -- synthesis translate_on
    port (DO : out STD_LOGIC_VECTOR (0 downto 0);
        ADDR : in STD_LOGIC_VECTOR (13 downto 0);
        CLK : in STD_ULOGIC;
        DI : in STD_LOGIC_VECTOR (0 downto 0);
        EN : in STD_ULOGIC;
        SSR : in STD_ULOGIC;
        WE : in STD_ULOGIC);
    end component;
```

Component declaration for BRAM (2)

```
component RAMB16_S9
    -- synthesis translate_off
    generic (
        INIT : bit_vector := X"0";
        INIT_00 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INIT_02 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INIT_03 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INIT_04 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INIT_05 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INIT_06 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INIT_07 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INITP_00 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        INITP_07 : bit_vector := X"0000000000000000000000000000000000000000000000000000000000000000";
        SRVAL : bit_vector := X"0";
        WRITE_MODE : string := "WRITE_FIRST";
    );
    -- synthesis translate_on
    port (DO : out STD_LOGIC_VECTOR (0 downto 0);
        DOP : out STD_LOGIC_VECTOR (1 downto 0);
        ADDR : in STD_LOGIC_VECTOR (13 downto 0);
        CLK : in STD_ULOGIC;
        DI : in STD_LOGIC_VECTOR (0 downto 0);
        DIP : in STD_LOGIC_VECTOR (0 downto 0);
        EN : in STD_ULOGIC;
        SSR : in STD_ULOGIC;
        WE : in STD_ULOGIC);
    end component;
```

General template of BRAM instantiation (1)

```
-- Component Attribute Specification for RAMB16_{S1 | S2 | S4}
-- Should be placed after architecture declaration but before the begin keyword
-- Put attributes, if necessary
-- Component Instantiation for RAMB16_{S1 | S2 | S4}
-- Should be placed in architecture after the begin keyword
RAMB16_{S1 | S2 | S4}_INSTANCE_NAME : RAMB16_S1
    -- synthesis translate_off
    generic map (
        INIT => bit_value,
        INIT_00 => vector_value,
        INIT_01 => vector_value,
        INIT_02 => vector_value,
        INIT_03 => vector_value,
        INIT_04 => vector_value,
        INIT_05 => vector_value,
        INIT_06 => vector_value,
        INIT_07 => vector_value,
        INIT_1F => vector_value,
        INITP_00 => vector_value,
        INITP_01 => vector_value,
        INITP_02 => vector_value,
        INITP_03 => vector_value,
        INITP_04 => vector_value,
        INITP_05 => vector_value,
        INITP_06 => vector_value,
        INITP_07 => vector_value,
        SRVAL => bit_value,
        WRITE_MODE => user_WRITE_MODE
    );
    -- synthesis translate_on
    port map (DO => user_DO,
        DOP => user_DOP,
        ADDR => user_ADDR,
        CLK => user_CLK,
        DI => user_DI,
        DIP => user_DIP,
        EN => user_EN,
        SSR => user_SSR,
        WE => user_WE);
```
Block RAM Waveforms – WRITE_FIRST mode

Block RAM Waveforms – READ_FIRST mode

Block RAM Waveforms – NO_CHANGE mode