

BRAM

There are two sources of memory available on most FPGA boards.

- Internal (on-chip memory)
- External SRAMs and DRAMs.

Internal memory is either **distributed** (from the LUTs) or **block** (dedicated on-chip memory).

Distributed is usually limited to Kbits, e.g., 30 K bits for the Spartan-3 XC3S200.

Using it also uses up resources used for normal logic.

Block allocated memory is limited in size (compared to external SRAM).

There are three ways to allocate memory in Xilinx chips

- HDL instantiation
- The CORE generator
- Behavioral HDL inference template

BRAM

Embedded block SRAM is surrounded by a synchronous configurable interface.

Each block consists of 16K bits plus an optional 2K parity bits.

It is organized in different widths, from 16K by 1 to 512 by 32.

The Spartan-3 XC3S200 has **12** block RAMs, for a total of 172K bits of storage.

The S3 board has 8M bits of external SRAM.

Both distributed and block RAM are already "wrapped" with a synchr. interface.

They are configurable to perform single- and dual-port access, and support various types of buffering and clocking schemes, enable, reset and initialization.

Because of these subtleties, it is difficult for the synthesis software to recognize the intention of the designer from the HDL code.

Therefore, HDL specified memory is not always portable.

Method 1: HDL Component Instantiation

Instantiating a Xilinx memory module does **not** include an HDL description for the architecture body.

The instantiation code for many Xilinx components can be obtained from ISE by selected *Edit, Language Templates* -- for a 16K-by-1 dual-port RAM:

```
-- RAMB16_S1_S1: Virtex-11/11-Pro,  
-- Spartan -3/3E 16k x 1 Dual-Port RAM  
-- Xilinx HDL Language Template version 8.1i  
RAMB16_S1_S1_inst: RAMB16_S1_S1  
  generic map(                                -- MUST CONSULT THE MANUAL  
    init_a => "0",                             -- TO GET INFO ON PARAMETERS  
    init_b => "0",  
    srval_a => "0",  
    srval_b => "0",  
    write_mode_a => "WRITE_FIRST",  
    write_mode_b => "WRITE_FIRST",  
    sim_collision_check => "ALL",  
    init_00 => x"0 ... 0",
```

Method 1: HDL Component Instantiation

```
        init_3f => x"0 ... 0"
    ),
    port map(
        doa => doa,      -- port a 1-bit data output
        dob => dob,      -- port b 1-bit data output
        addra => addra,  -- port a 14-bit address input
        addrb => addrb,  -- port b 14-bit address input
        clka => clka,    -- port a clk
        clkb => clkb,    -- port b clk
        dia => dia,      -- port a 1-bit data input
        dib => dib,      -- port b 1-bit data input
        ena => ena,      -- port a ram enable input
        enb => enb,      -- port b ram enable input
        ssra => ssra,    -- port a synch. set/reset input
        ssrb => ssrb,    -- port b synch. set/reset input
        wea => wea,      -- port a write enable input
        web => web,      -- port b write enable input
    );
```

Method 2: CORE generator

To simplify the instantiation process, Xilinx provides a utility program called the CORE generator.

Invoke it using *Project, New Source* and select IP (Coregen & Architecture Wizard) in the Wizard dialog.

The CORE generator program guides you through a series of questions and then generates several files.

The file with the *.xco* extension is a text file that contains the information needed to construct the desired memory component.

The file with the *.vhd* extension contains the "wrapper" code for simulation purposes.

Although *Coregen* is more convenient than direct HDL instantiation, it is **not** within the HDL framework and can lead to comparability problems outside the ISE env.

Method 3: Memory Module via HDL Inference

It is not possible to write a **device-independent** HDL description of a memory module.

However, XST (the Xilinx synthesis program) provides a collection of behavioral HDL templates to *infer* memory modules from Xilinx FPGA devices.

The templates are implemented as *behavioral descriptions* and contain **no** device specific component instantiation.

Be aware that the code may not be recognized by other third-party synthesis software.

Therefore, the templates are *semi-portable* behavioral descriptions.

Also, the template approach is based on XST being able to recognize the template and infer the proper memory module accordingly.

Updates to the software may cause some code to be mis-interpreted.

Check the synthesis report to be sure the desired memory module is inferred.

Method 3: Memory Module via HDL Inference**Single-port RAM**

The embedded memory of a Spartan-3 device is wrapped with a synchronous interface.

During a **write** operation, at the rising edge of the clock, the **address**, **input data**, and *relevant control signals*, such as **we** (write enable) are sampled.

If **we** is asserted, a write operation is performed.

The **read** operation can be synchronous or asynchronous.

For asynchronous, the address signal is used directly to access the RAM array (can only be used with *distributed* RAM).

After the address changes, the data becomes available after a short delay.

For synchronous, the address is sampled on the rising edge of clk and stored in a register.

The data becomes available following the clk and a short delay.

Method 3: Memory Module via HDL Inference

```
-- single-port RAM with synchronous read
-- modified from XST 8.1i rams_07
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity xilinx_one_port_ram_sync is
    generic(
        ADDR_WIDTH: integer := 12;
        DATA_WIDTH: integer := 8;
    );
    port(
        clk: in std_logic;
        we: in std_logic;
        addr: in std_logic_vector(ADDR_WIDTH-1 downto 0);
        din: in std_logic_vector(DATA_WIDTH-1 downto 0);
        dout: out std_logic_vector(DATA_WIDTH-1 downto 0)
    );
end xilinx_one_port_ram_sync;
```


Method 3: Memory Module via HDL Inference

```
architecture beh_arch of xilinx_one_port_ram_sync is
  type ram_type is array (2**ADDR_WIDTH-1 downto 0)
    of std_logic_vector (DATA_WIDTH-1 downto 0);
  signal ram: ram_type;
  signal addr_reg: std_logic_vector(ADDR_WIDTH-1 downto
0);
begin
  process (clk)
    begin
      if ( clk'event and clk = '1') then
        if (we = '1') then
          ram(to_integer(unsigned(addr))) <= din;
        endif;
        addr_reg <= addr;
      endif;
    end process;
    dout <= ram(to_integer(unsigned(addr_reg)));
end beh_arch;
```

Method 3: Memory Module via HDL Inference

Here, we see *addr* is sampled and stored to *addr_reg* register on the rising edge of *clk*.

Also, the memory array (*ram* signal) is addressed via the *addr_reg* signal.