VHDL
Memórias

4 de outubro de 2015

Rafael Corsi Ferrão - IMT

rafael.corsi@maua.br
http://www.maua.br

CONTEÚDO

CONTEÚDO

MEMÓRIAS

Dois tipos distintos de memórias podem ser usados em FPGA:

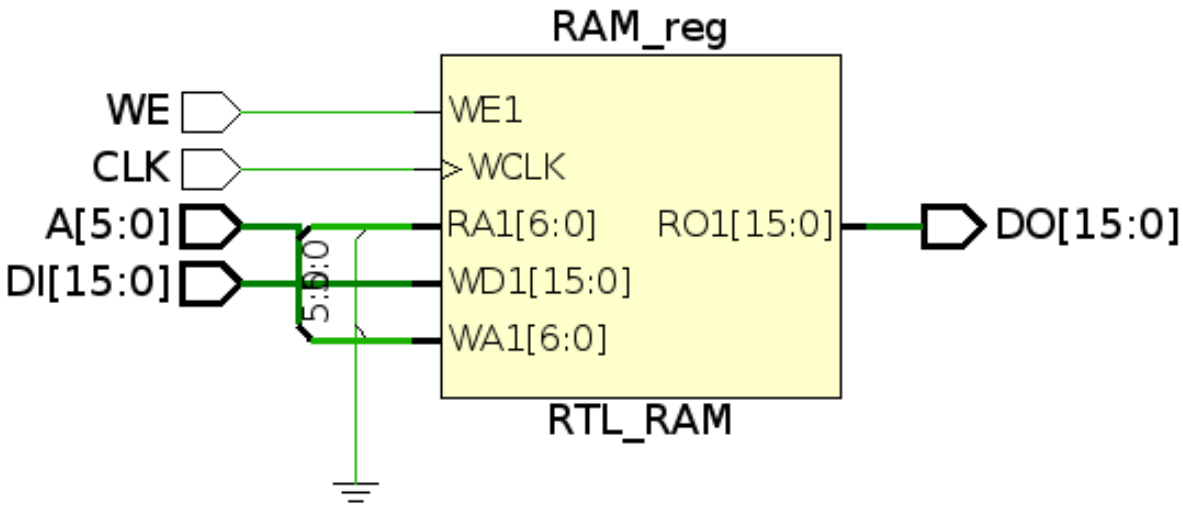
- Memória interna
  - RAM
  - ROM
  - FIFO
- Memória externa
  - SRAM
  - DDR2, DDR3
  - Flash

No caso de memórias externas em alguns casos, faz-se necessário a utilização de um controlador de memória para gerenciar o acesso atualização, etc. Para o controle e acesso das memórias externas é aconselhável a utilização de IP Cores já validados e otimizados

CONTEÚDO

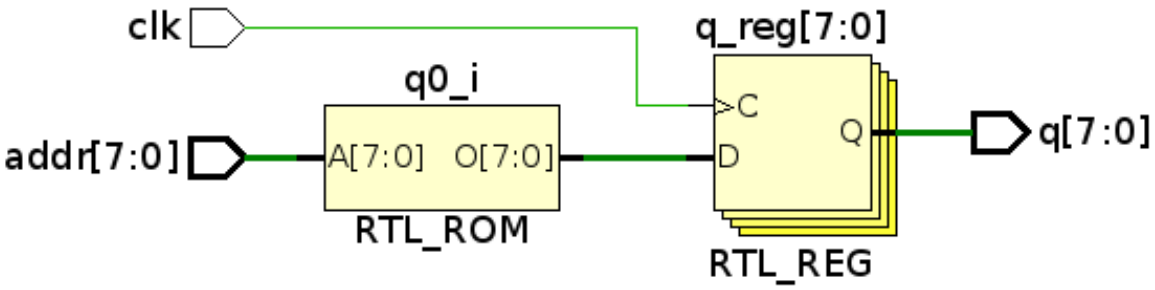
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity rams_04 is
    port (CLK : in std_logic;
          WE  : in std_logic;
          A   : in std_logic_vector(5 downto 0);
          DI  : in std_logic_vector(15 downto 0);
          DO  : out std_logic_vector(15 downto 0));
end rams_04;

architecture syn of rams_04 is

    type ram_type is array (63 downto 0) of std_logic_vector (15 downto 0);
    signal RAM : ram_type;

begin

    process (CLK)
    begin
        if (CLK'event and CLK = '1') then
            if (WE = '1') then
                RAM(conv_integer(A)) <= DI;
            end if;
        end if;
    end process;

    DO <= RAM(conv_integer(A));

end syn;
```

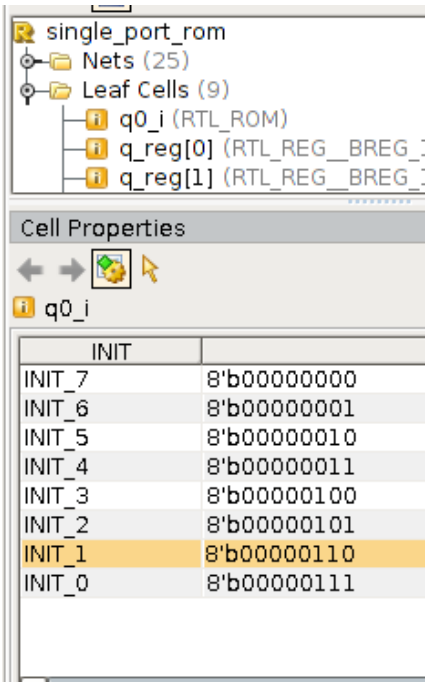
RAM RTL


RAM_reg
WE
CLK
A[5:0]
DI[15:0]
5:0
WE1
WCLK
RA1[6:0]
RO1[15:0]
WD1[15:0]
WA1[6:0]
RTL_RAM
DO[15:0]

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

entity single_port_rom is
        port(
          addr        : in natural range 0 to 255;
          clk                 : in std_logic;
          q                   : out std_logic_vector(7 downto 0)
        );

end entity;

architecture rtl of single_port_rom is

        -- Build a 2-D array type for the RoM
        type memory_t is array(7 downto 0) of std_logic_vector(7 downto 0);

        -- Declare the ROM signal and specify a default value.
        signal rom : memory_t := (X"00", X"01", X"02", X"03", X"04", X"05", X"06", X"07");

begin

  process(clk)
    begin
      if(rising_edge(clk)) then
          q <= rom(addr);
      end if;
    end process;

end rtl;
```

ROM RTL


clk
q_reg[7:0]
q0_i
addr[7:0]
A[7:0]
O[7:0]
C
D
Q
q[7:0]
RTL_ROM
RTL_REG


single_port_rom
Nets (25)
Leaf Cells (9)
q0_i (RTL_ROM)
q_reg[0] (RTL_REG__BREG_
q_reg[1] (RTL_REG__BREG_
Cell Properties
q0_i
INIT
INIT_7 8'b00000000
INIT_6 8'b00000001
INIT_5 8'b00000010
INIT_4 8'b00000011
INIT_3 8'b00000100
INIT_2 8'b00000101
INIT_1 8'b00000110
INIT_0 8'b00000111

FIFO

Usar um IP Core para FIFO

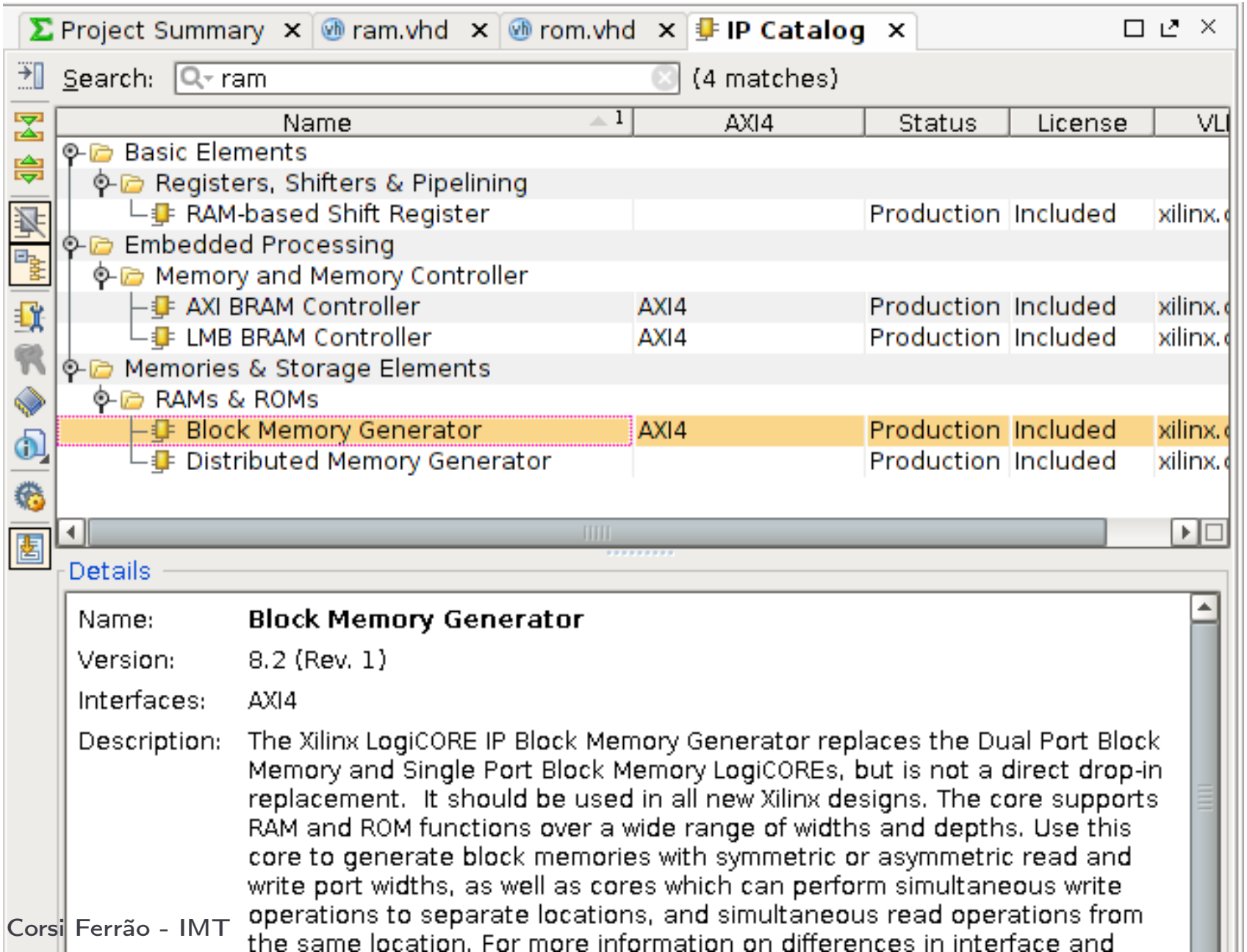
CONTEÚDO

MEMÓRIA DISTRIBUIDA VS BLOCO

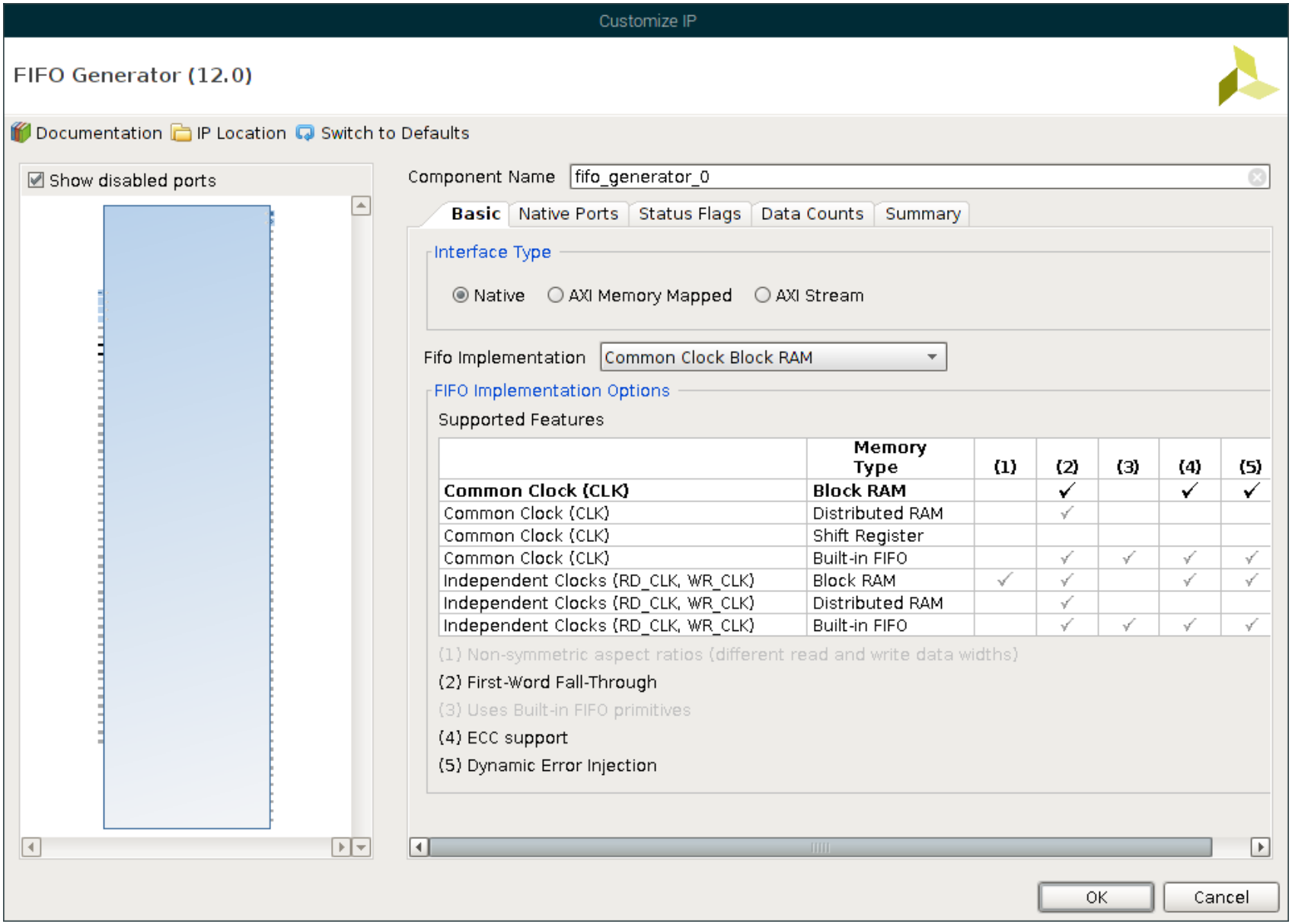
Podemos usar memórias distribuídas (formadas por registradores) ou memórias em blocos (dedicadas) para sintetizar os tipos de memórias, as diferenças são:

- Distribuídas
  - Mais rápidas
  - Fazem uso de registradores para armazenar as informações
  - Maior consumo de área
- Bloco
  - Memórias dedicadas
  - Não fazem uso de registradores que poderiam ser usados em outras lógicas
  - Mais lentos que as distribuídas

IP CORE

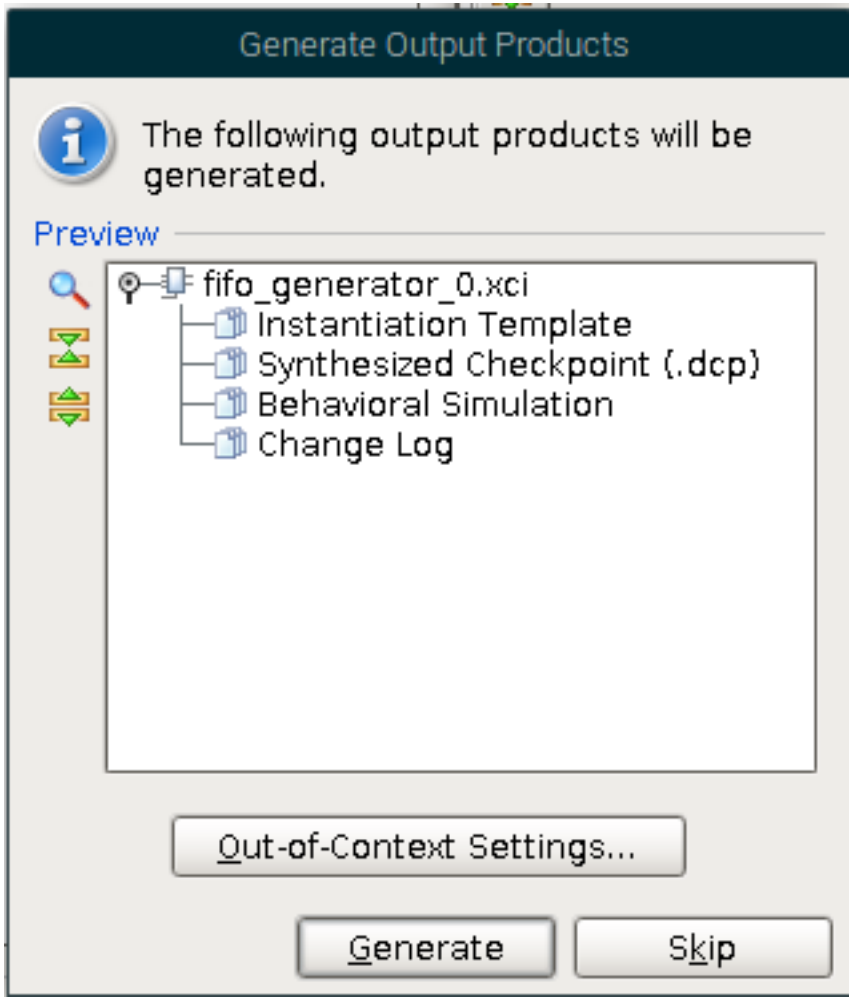

Project Summary
ram.vhd
rom.vhd
IP Catalog
Search: ram
(4 matches)
Name
AXI4
Status
License
Basic Elements
Registers, Shifters & Pipelining
RAM-based Shift Register
Production
Included
Embedded Processing
Memory and Memory Controller
AXI BRAM Controller
AXI4
Production
Included
LMB BRAM Controller
AXI4
Production
Included
Memories & Storage Elements
RAMs & ROMs
Block Memory Generator
AXI4
Production
Included
Distributed Memory Generator
Production
Included
Details
Name: Block Memory Generator
Version: 8.2 (Rev. 1)
Interfaces: AXI4
Description: The Xilinx LogiCORE IP Block Memory Generator replaces the Dual Port Block Memory and Single Port Block Memory LogiCOREs, but is not a direct drop-in replacement. It should be used in all new Xilinx designs. The core supports RAM and ROM functions over a wide range of widths and depths. Use this core to generate block memories with symmetric or asymmetric read and write port widths, as well as cores which can perform simultaneous write operations to separate locations, and simultaneous read operations from the same location. For more information on differences in interface and

FIFO


Customize IP
FIFO Generator (12.0)
Documentation
IP Location
Switch to Defaults
Show disabled ports
Component Name
fifo_generator_0
Basic
Native Ports
Status Flags
Data Counts
Summary
Interface Type
Native
AXI Memory Mapped
AXI Stream
Fifo Implementation
Common Clock Block RAM
FIFO Implementation Options
Supported Features
Memory Type
(1)
(2)
(3)
(4)
(5)
Common Clock (CLK) Block RAM ✓ ✓ ✓
Common Clock (CLK) Distributed RAM ✓
Common Clock (CLK) Shift Register
Common Clock (CLK) Built-in FIFO ✓ ✓ ✓ ✓
Independent Clocks (RD_CLK, WR_CLK) Block RAM ✓ ✓ ✓ ✓
Independent Clocks (RD_CLK, WR_CLK) Distributed RAM ✓
Independent Clocks (RD_CLK, WR_CLK) Built-in FIFO ✓ ✓ ✓ ✓
(1) Non-symmetric aspect ratios (different read and write data widths)
(2) First-Word Fall-Through
(3) Uses Built-in FIFO primitives
(4) ECC support
(5) Dynamic Error Injection
OK
Cancel

COMO USAR ?

Quando gerado um IP Core pela interface do Vivado, os seguintes itens são adicionados ao projeto :



FIFO ENTIDADE GERADA

```
ENTITY fifo_generator_0 IS
  PORT (
    clk : IN STD_LOGIC;
    rst : IN STD_LOGIC;
    din : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
    wr_en : IN STD_LOGIC;
    rd_en : IN STD_LOGIC;
    dout : OUT STD_LOGIC_VECTOR(17 DOWNTO 0);
    full : OUT STD_LOGIC;
    empty : OUT STD_LOGIC
  );
END fifo_generator_0;
```