INTRODUCTION TO FPGA PROGRAMMING

LESSON 07: STORING DATA ON FPGAS: RAMS AND FIFOS

Dr. Davide Cieri[1]

[1]Max-Planck-Institut für Physik, Munich

September 2024

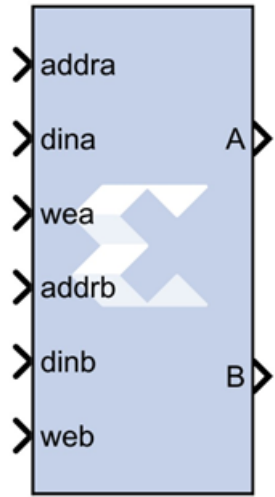
MAX-PLANCK-INSTITUT FÜR PHYSIK

STORING DATA ON FPGAS

- In previous lectures, we saw that Flip-Flops are a solution to store data on FPGA.
  - Good to store a small amount of data.
  - Ideal for holding state information and temporary data.
- If you want to store more data, a RAM is a better choice
- If very large data storage is needed, FPGAs can interface with external memory (DRAM, SRAM, Flash, etc.)

WHAT'S A RAM?

- Random-Access Memory (RAM) is a common block that allows you to store data within an FPGA, and read it back later
- *Random-Access* means that data can be accessed in any order
  - One clock you read at location 1, the next at location 32
- RAMs are typically single or dual-port
  - Single-port means in one clock cycle you can either write or read to/from the RAM, but not together
  - Dual-port RAM have separate ports that allows you to read and write at the same time

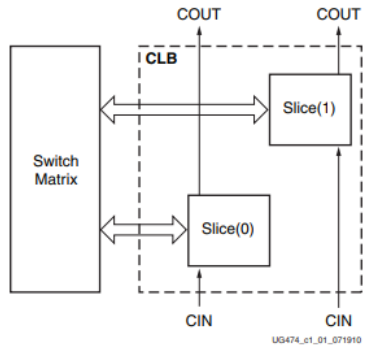


RAM ON FPGAS

- AMD FPGA logic resources are distributed in *Configurable Logic Blocks (CLBs)* and *Slices*
- Logic resources can be assembled together to create small RAM blocks (Distributed RAM)
- FPGAs typically have on-chip memory blocks that can be used of larger data storage (Block RAM)
- Both Distributed and Block RAM supports the implementation of single and dual port RAM blocks

CLBS

- CLBs are the main logic resources for implementing both sequential and combinatorial circuits.
- 7-series CLBs contains two slices, independently connected to the switch matrix for general routing
  - No direct connection between slices



SLICES

- Each Logic Slice consists of
  - 4 LUT6 (six inputs)
  - 8 Flip-Flops
  - Wide-function multiplexer
  - Carry logic
- Two types of Logic Slices available
  - SLICEL for logic arithmetic and ROM functions
  - SLICEM additionally can be configured to store data as distributed RAM or shift register
- The Artix-7A35T has 3600 SLICEL and 1600 SLICEM
- Maximum allowed Distributed RAM of 400Kb
- Full description of Xilinx CLBs here

SLICEL AND SLICEM

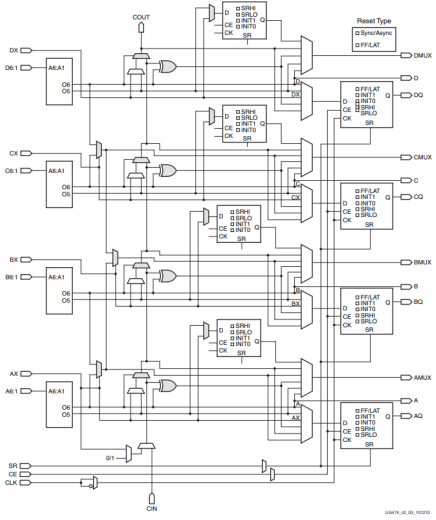


Figure 2-4: **Diagram of SLICEL**

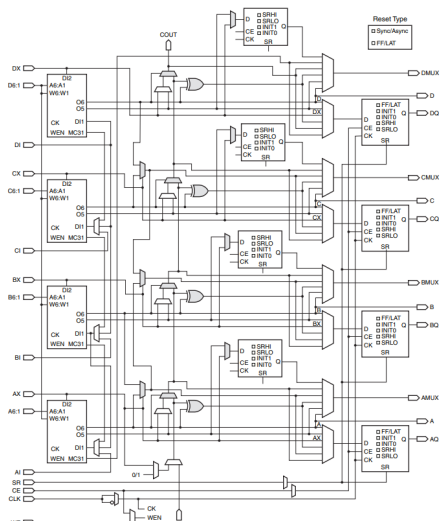


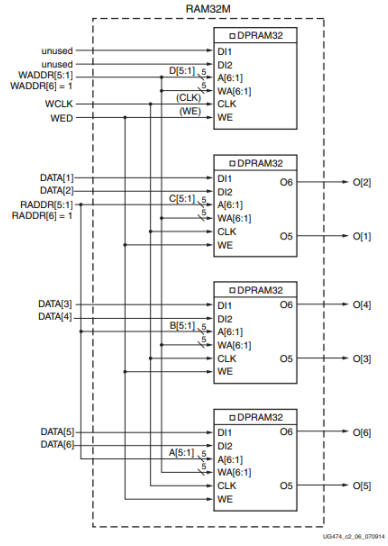
Figure 2-3: **Diagram of SLICEM**

SLICE CONTROL AND CLOCK SIGNALS

- Clock, resets and enable signals are common to a slices
- Elements with different control signals cannot be in the same slices
- All elements in a slice react to the same clock edge

DISTRIBUTED RAM CONFIGURATION

- Slice-M can be configured as a synchronous RAM resource called distributed RAM
- Distributed RAMs have synchronous write and asynchronous read ports
- Possible configurations
  - Single Port - Common Address for read and write (32×1 bit)
  - Simple Dual Port - Separate port addresses for read and write (64×6 bit)
  - Dual Port and Quad Port



BLOCK RAM

- Block RAMs are dedicated storage components built in the FPGA.
- In contrast to distributed RAM, they can implement a true dual-port RAM
  - Both write and read port are synchronous and have independent clocks
  - Contention might be an issue (WRITE or READ first)
- BRAM can be configured as 1×72 Kb or 2×36 Kb
- Artix XC7A35Y has 1800 BRAM blocks available

RAMS IN VHDL

- A RAM module is instantiated in VHDL by defining an `array` type
- Vivado automatically choose the RAM type (distributed, block) depending on the size of the array
- RAM functionalities (single, dual, quad, ROM, FIFO) are defined with the architecture and port definitions
- AMD examples available here.

```
-- Distributed RAM instantiation
type distributed_ram_type is array (63 downto 0) of std_logic_vector(15 downto 0);
signal DRAM : distributed_ram_type;
-- Block RAM instantiation
type block_ram_type is array (1023 downto 0) of std_logic_vector(15 downto 0);
signal BRAM : block_ram_type;
```

FORCE THE RAM TYPE

- You can force the synthesiser to use a particular RAM type using the `attribute` property
- We met attribute in the past lectures. They are additional information of a particular VHDL type
- You can define new attributes and attach them to any type
- Some of these attributes are parsed by Vivado for the synthesis
- A list of attributes that can be used in Xilinx FPGAs is available here.
- To define the RAM type use the `RAM_STYLE` attribute
  - Allowed values are: `distributed, block, registers, ultra, mixed` and `auto`

```
attribute ram_style : string;
attribute ram_style of myram : signal is "distributed";
```

EXAMPLE: SINGLE-PORT DISTRIBUTED RAM WITH ASYNCHRONOUS READ

- https://docs.amd.com/r/en-US/ug901-vivado-synthesis/Single-Port-RAM-with-Asynchronous-Read-Coding-Example-VHDL

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

entity rams_dist 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_dist;
```

```vhdl
architecture syn of rams_dist 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 rising_edge(clk) then
      if (we = '1') then
        RAM(to_integer(unsigned(a))) <= di;
      end if;
    end if;
  end process;

  do <= RAM(to_integer(unsigned(a)));

end syn;
```

EXAMPLE: DUAL-PORT BRAM WITH SINGLE CLOCK

```
entity dual_port_bram is
Port (
  data_in_a : in STD_LOGIC_VECTOR(7 downto 0);
  data_in_b : in STD_LOGIC_VECTOR(7 downto 0);
  rw_a : in STD_LOGIC;
  rw_b : in STD_LOGIC;
  clk : in STD_LOGIC;
  address_a : in STD_LOGIC_VECTOR(9 downto 0);
  address_b : in STD_LOGIC_VECTOR(9 downto 0);
  data_out_a : out STD_LOGIC_VECTOR(7 downto 0)
       ;
  data_out_b : out STD_LOGIC_VECTOR(7 downto 0)
);
end dual_port_bram;
```

```
architecture Behavioral of dual_port_ram is
  type ram_type is array(0 to 1023) of std_logic_vector(7 downto 0);
  signal RAM : ram_type;
begin
  input_a : process (clk)
    begin
      if rising_edge(clk) then
        if rw_a = '1' then
          RAM(to_integer(unsigned(address_a))) <= data_in_a;
          end if;
        if rw_b = '1' then
          RAM(to_integer(unsigned(address_b))) <= data_in_b;
        end if;
        data_out_b <= RAM(to_integer(unsigned(address_b)));
        data_out_a <= RAM(to_integer(unsigned(address_a)));
      end if;
    end process;
end Behavioral;
```

ROMS

- RAMs on FPGAs can also be configured as ROMs (Read-Only-Memory)
  - No write port
  - Content of the RAM must be initialised
  - Also normal RAMs can be initialised in the same way

```
...
type romtype is array(0 to 15) of std_logic_vector(7 downto 0);
signal memory_ram : romtype := (
X"00", X"01", X"02", X"03", X"04", X"05", X"06", X"07",
X"08", X"09", X"0A", X"0B", X"0C", X"0D", X"0E", X"0F");

begin
  process (clk)
  begin
    if(rising_edge(clk)) then
      data_out <= memory_ram(to_integer(unsigned(address)));
    end if;
  end process;
end Behavioral;
```

INITIALISE RAMS FROM FILE

- In VHDL you can initialise the RAM content from an external file
  - External file can be of any type (`.txt, .dat`, etc...)
  - Each line describes the initial content at an address position in the RAM
  - Number of lines must be equal to RAM depth
  - RAM content can be expressed in hexadecimal or binary (not both together!)
  - No other content allowed in the file (no comments)

```
type RamType is array(0 to 7) of bit_vector(31 downto 0);
impure function InitRamFromFile (RamFileName : in string) return RamType is
  FILE RamFile : text is in RamFileName;
  variable RamFileLine : line;
  variable RAM : RamType;
begin
  for I in RamType'range loop
    readline (RamFile, RamFileLine);
    read (RamFileLine, RAM(I));
  end loop;
  return RAM;
end function;
signal RAM : RamType := InitRamFromFile("myfile.txt");
```

FIFOS

- The First-In, First Out (FIFO) is a common block to store data
- Data comes in one entry at the time and gets read out from oldest to newest (no address port required)
  - Don't write to a full FIFO (overflow), don't read from an empty FIFO (underflow)
  - `full` and `empty` ports signal the status of the memory
  - Optional additional ports to check FIFO status (`almost_full, almost_empty`)
- It can be implemented either as distributed or block RAMs


```
entity FIFO is
port (
    clk        : in  std_logic;
    reset      : in  std_logic;
    write_en   : in  std_logic;
    read_en    : in  std_logic;
    data_in    : in  std_logic_vector(7 downto 0);
    data_out   : out std_logic_vector(7 downto 0);
    full       : out std_logic;
    empty      : out std_logic
);
end FIFO;
```

EXAMPLE: FIFO

```
architecture Behavioral of FIFO is
  type fifo_array is array (15 downto 0) of
       std_logic_vector(7 downto 0);
  signal fifo_mem : fifo_array := (others => (others
       => '0'));
  signal write_ptr, read_ptr : integer range 15
       downto 0 := 0;
  signal count     : integer range 16 downto 0 := 0;
begin
  process(clk, reset)
    begin
      if reset = '1' then
        write_ptr <= 0;
        read_ptr <= 0;
        count <= 0;
        fifo_mem <= (others => (others => '0'));
```

```
...
        elsif rising_edge(clk) then
            -- Write operation
            if write_en = '1' and full = '0' then
                fifo_mem(write_ptr) <= data_in;
                write_ptr <= write_ptr + 1;
                count <= count + 1;
            end if;
            -- Read operation
            if read_en = '1' and empty = '0' then
                data_out <= fifo_mem(read_ptr);
                read_ptr <= read_ptr + 1;
                count <= count - 1;
            end if;
        end if;
  end process;
  -- Full and empty flags
  full <= '1' when count = 16 else '0';
  empty <= '1' when count = 0 else '0';
end Behavioral;
```

LAB 11: TRIGONOMETRIC FUNCTION ON FPGAS

The figures in these slides are taken from:

- Digital Design: Principles and Practices, Fourth Edition, John F. Wakerly, ISBN 0-13- 186389-4.

- allaboutfpga.com
- nandland.com
- docs.amd.com
- https://www.symmetryelectronics.com/
- https://www.edn.com/