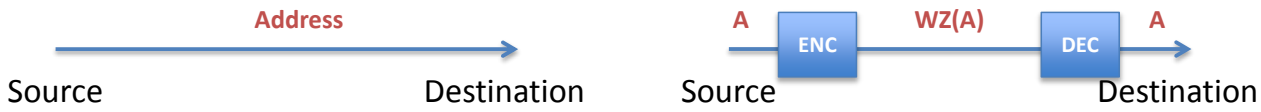
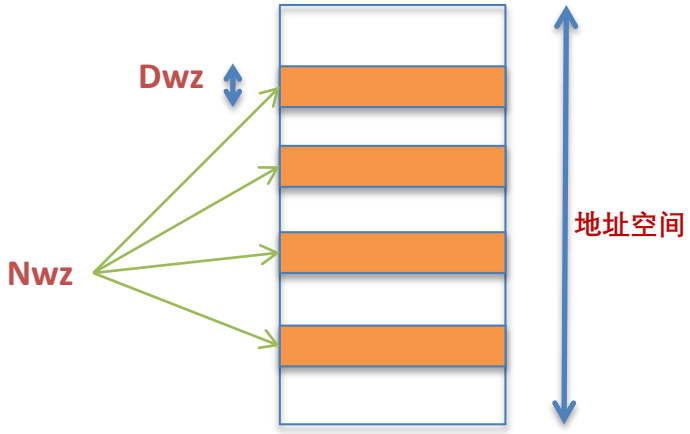
启发与“working zone”的低功耗编码方法。

Working zone 编码方法是为降低地址总线设计的编码方法：如果地址属于某个间隔（称为 **working zone**），则用于转换地址的数值。



Working zone 定义为，从基地址开始，拥有固定大小的地址范围（**Dwz**）。编码方法中可存在多个 working zone（**Nwz**）。



修改后的编码方案如下：

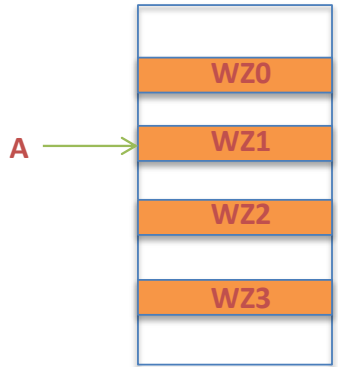
- 如果要发送的地址（**ADDR**）不属于任何 working zone 它则按原样发送，并且，一个相对于寻址 bit 的附加 bit（**WZ_BIT**）将被设置为 0。

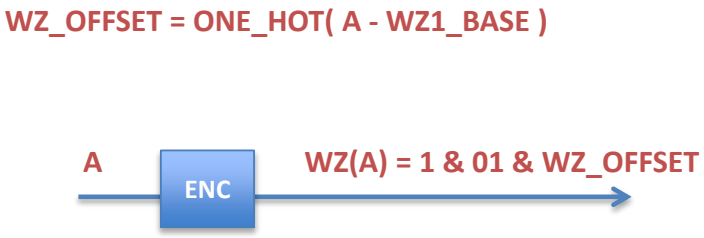
  总的来说 给予一个 ADDR 后，将发送与 ADDR 串联的 WZ_BIT = 0（**WZ_BIT&ADDR**，其中 & 是串联的符号）；

- 如果要发送的地址（**ADDR**）属于 working zone，则将附加位（**WZ_BIT**）设置为 1，而地址的 bit 则分为 2 段，分别表示：
  - 地址所属的 working zone 编号（**WZ_NUM**），以二进制编码。
  - 相对于 working zone 的基址的偏移量（**WZ_OFFSET**），以 one-hot 编码（要表示的值等效于编码唯一的 1 位）。

SYMBOL	*BINARY*	*ONE-HOT*
0	000	00000001
1	001	00000010
2	010	00000100
3	011	00001000
4	100	00010000
5	101	00100000
6	110	01000000
7	111	10000000

简单来说，给予一个 ADDR 后，将发送与 WZ_NUM 和 WZ_OFFSET 串联的 WZ_BIT = 1（**WZ_BIT& WZ_NUM& WZ_OFFSET**，其中 & 是串联的符号）；





<u>在要实现的版本中</u>：需考虑的待编码地址 ADDR 的位数为 7，意味着从 0 到 127 的那些地址将定义为有效地址。working zone 的数量为 8（Nwz = 8），而大小，包括基本地址，为 4 个地址（Dwz = 4）。这意味着已编码的地址将由 8 位组成：WZ_BIT 的 1 位+ADDR 的 7 位，或 WZ_BIT 的 1 位 + 该地址所属的 working zone 编号的二进制的 3 位 + 相对于基址的 ADDR 的偏移值的 one-hot 编码 4 位。

要实现的模块需读取要编码的地址和 working zone 的 8 个基地址，并且须产生适当的编码地址。

数据:

每个 8 位大小的数据都存储在存储器中，并从位置 0 开始对字节进行寻址。指定为 7 位的地址也存储在 8 位中。 第八位值将始终为零。

- 从 0 到 7 的存储位置用于存储 working zone 的八个基地址：

  0 - 基地址 WZ 0

  1 - 基地址 WZ 1

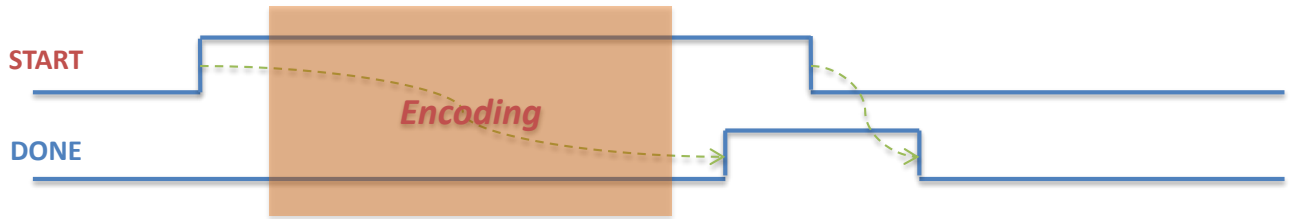
  …..

  7 - 基地址 WZ 7

- 存储器中的位置 8 包含需编码的值（地址）（ADDR）；
- 存储器中的位置 9 用于最后输入根据先前规则编码的值

其他说明:

1. 在 ONE-HOT 编码中，将位 0 视为最低有效位。
   - WZ_OFFSET = 0　　one hot 0001;
   - WZ_OFFSET = 1　　one hot 0010;
   - WZ_OFFSET = 2　　one hot 0100;
   - WZ_OFFSET = 3　　one hot 1000;
2. 再次强调，编码后的地址将如下组成：
   - Bit 7: WZ_BIT（0 或 1）；
   - Bit 6-4: WZ_NUM（二进制）；
   - Bit 3-0: WZ_OFFSET（one-hot）
3. 如有必要，请考虑 working zone 的基址在同一执行中永远不会改变；
4. 当输入的 START 信号升高到 1 时，模块将开始处理。START 信号将保持高电平，直到 DONE 信号升高到高电平为止。 在计算结束时（将结果写入存储器后），要设计的模块必须将 DONE 信号升高（变为 1）通知处理结束。 在 START 信号重置为 0 之前 DONE 信号必须保持高电平。在 DONE 重置为 0 之前无法提供新的 START 信号。如果此时 START 信号升高到 1，则模块必须重新启动编码阶段。



示例:

以下数字序列显示了处理后的存储器内容示例。 此处以十进制表示的值将以二进制编码存储在内存中的 8 个 (无符号) 位。

CASE 1: 指数不存在于任何 working zone

存储器地址	数值	
0	4	//基地址 WZ 0
1	13	//基地址 WZ 1
2	22	//基地址 WZ 2
3	31	//基地址 WZ 3
4	37	//基地址 WZ 4
5	45	//基地址 WZ 5
6	77	//基地址 WZ 6
7	91	//基地址 WZ 7
8	42	//需编码的 ADDR
9	42	//编码后 output

CASE 2: 指数存在于一个 working zone

存储器地址	数值	
0	4	//基地址 WZ 0
1	13	//基地址 WZ 1
2	22	//基地址 WZ 2
3	31	//基地址 WZ 3
4	37	//基地址 WZ 4
5	45	//基地址 WZ 5
6	77	//基地址 WZ 6
7	91	//基地址 WZ 7
8	33	//需编码的 ADDR
9	180	//编码后 output (1-011-0100)

组件界面：

```
entity project_reti_logiche is
      port (
            i_clk             : in  std_logic;
            i_start           : in  std_logic;
            i_rst             : in  std_logic;
            i_data            : in  std_logic_vector(7 downto 0);
            o_address         : out std_logic_vector(15 downto 0);
            o_done            : out std_logic;
            o_en              : out std_logic;
            o_we              : out std_logic;
            o_data            : out std_logic_vector (7 downto 0)
      );
end project_reti_logiche;
```

i_clk	TestBench 生成的 CLOCK 入口信号
i_start	TestBench 生成的 START 信号
i_rst	是用于初始化机器以准备接收第一个 start 信号的 RESET 信号
i_data	请求读取后从存储器到达的信号
o_address	是地址发送到存储器的输出信号
o_done	是用于传达处理结束和输出数据写入存储器的输出信号
o_en	是为进行通信而发送到存储器的 ENABLE 信号 （读取与写入）
o_we	是为写入而要发送到存储器（= 1）的 WRITE ENABLE 信号。 若是为了读取则须等于 0
o_data	是从组件到存储器的输出信号。

存储器已在 TEST BENCH 中实例化

可以从以下 VHDL 中提取存储器，该描述是 TEST BENCH 的一部分，源于以下链接《VIVADO 用户指南》：

https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_3/ug901-vivado- synthesis.pdf

```
--   Single-Port   Block   RAM   Write-First   Mode   (recommended   template)
--
--                              File:                              rams_02.vhd
--
library                                                                  ieee;
use                                              ieee.std_logic_1164.all;
use                                            ieee.std_logic_unsigned.all;
entity                               rams_sp_wf                            is
port(
  clk                        :           in                        std_logic;
  we                               :          in                   std_logic;
  en                               :          in                   std_logic;
  addr       :       in             std_logic_vector(15       downto       0);
  di                 :      in           std_logic_vector(7      downto      0);
  do                   :       out       std_logic_vector(7       downto       0)
);
end                                                                rams_sp_wf;

architecture            syn            of            rams_sp_wf            is
type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);
signal                    RAM                    :                    ram_type;
begin
  process(clk)
    begin
    if        clk'event        and        clk        =        '1'        then
      if               en               =               '1'               then
        if               we               =               '1'               then
          RAM(conv_integer(addr))                    <=                    di;
          do                                                        <=   di;
        else
          do                   <=                  RAM(conv_integer(addr));
        end                                                                 if;
      end                                                                   if;
    end                                                                     if;
  end                                                                  process;
end                                                                        syn;
```