1. Kmap1
   * library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity kmapone is

Port ( A : in STD\_LOGIC;

B : in STD\_LOGIC;

C : in STD\_LOGIC;

D : in STD\_LOGIC;

output : out STD\_LOGIC);

end kmapone;

architecture Behavioral of kmapone is

signal tmp: std\_logic\_vector (3 downto 0);

begin

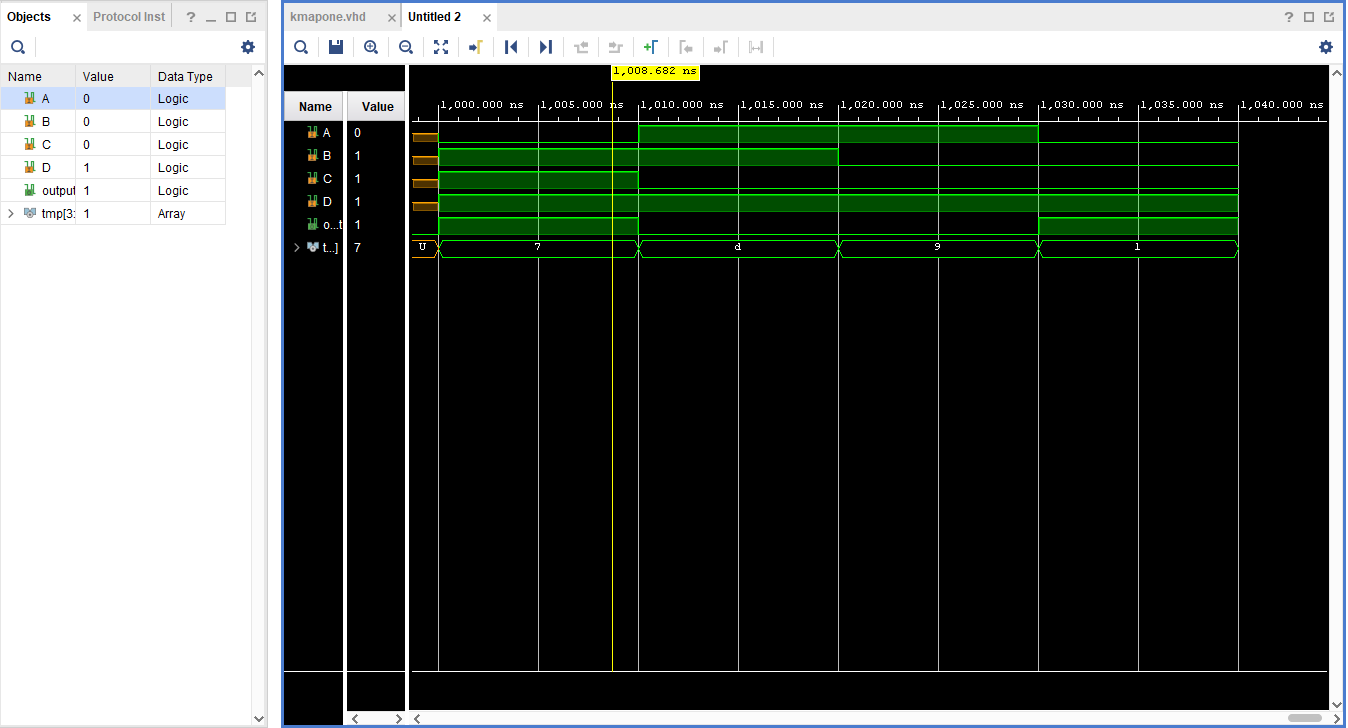
tmp <= A&B&C&D;

with tmp select

output<='1' when "0---",

'0' when others;

end Behavioral;

* + 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ABCD | 00 | 01 | 11 | 10 |
| 00 | 1 | 1 | 0 | 0 |
| 01 | 1 | 1 | 0 | 0 |
| 11 | 1 | 1 | 0 | 0 |
| 10 | 1 | 1 | 0 | 0 |

F= A’

1. Kmap2
   * library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity kmaptwo is

Port ( A : in STD\_LOGIC;

B : in STD\_LOGIC;

C : in STD\_LOGIC;

D : in STD\_LOGIC;

output : out STD\_LOGIC);

end kmaptwo;

architecture Behavioral of kmaptwo is

signal tmp: std\_logic\_vector (3 downto 0);

begin

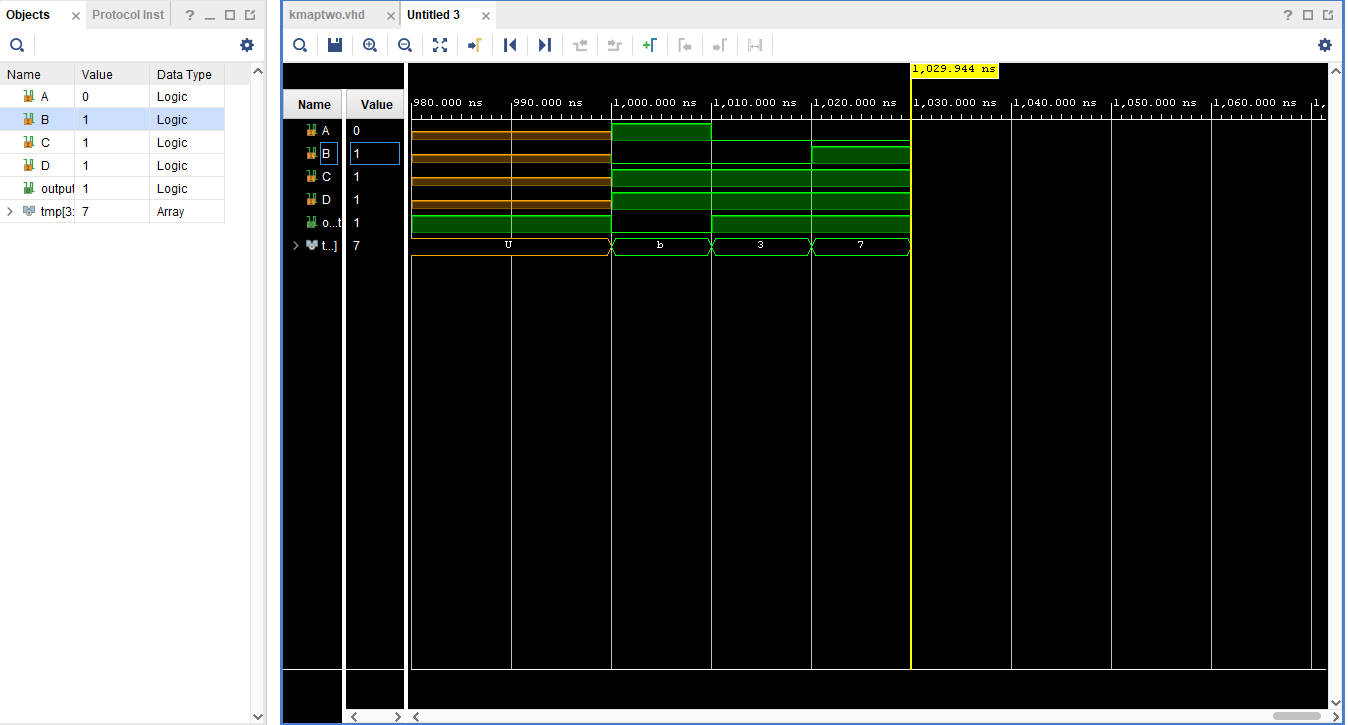
tmp <= A&B&C&D;

with tmp select

output<='0' when "1---",

'1' when others;

end Behavioral;

* + 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ABCD | 00 | 01 | 11 | 10 |
| 00 | 1 | 1 | 0 | 0 |
| 01 | 1 | 1 | 0 | 0 |
| 11 | 1 | 1 | 0 | 0 |
| 10 | 1 | 1 | 0 | 0 |

* + F=A
  + Will output 0 because of product of sums

1. Kmap
   * The first bit will represent the mode, while the 3 after are the 3 bit number that will be evaluated. The output needs to be 5 bits long. Each output will have its own kmap.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Mode | A | | B | | C | Out-4 | | Out-3 | | Out-2 | Out-1 | | Out-0 |
| 0 | 0 | | 0 | | 0 | 0 | | 0 | | 0 | 0 | | 0 |
| 0 | 0 | | 0 | | 1 | 0 | | 0 | | 0 | 1 | | 1 |
| 0 | 0 | | 1 | | 0 | 0 | | 0 | | 1 | 1 | | 0 |
| 0 | 0 | | 1 | | 1 | 0 | | 1 | | 0 | 0 | | 1 |
| 0 | 1 | | 0 | | 0 | 0 | | 1 | | 1 | 0 | | 0 |
| 0 | 1 | | 0 | | 1 | 0 | | 1 | | 1 | 1 | | 1 |
| 0 | 1 | | 1 | | 0 | 1 | | 0 | | 0 | 1 | | 0 |
| 0 | 1 | | 1 | | 1 | 1 | | 0 | | 1 | 0 | | 1 |
| 1 | 0 | | 0 | | 0 | 0 | | 0 | | 0 | 0 | | 0 |
| 1 | 0 | | 0 | | 1 | 0 | | 0 | | 0 | 0 | | 1 |
| 1 | 0 | | 1 | | 0 | 0 | | 0 | | 0 | 0 | | 1 |
| 1 | 0 | | 1 | | 1 | 0 | | 0 | | 0 | 1 | | 0 |
| 1 | 1 | | 0 | | 0 | 0 | | 0 | | 0 | 0 | | 1 |
| 1 | 1 | | 0 | | 1 | 0 | | 0 | | 0 | 1 | | 0 |
| 1 | 1 | | 1 | | 0 | 0 | | 0 | | 0 | 1 | | 0 |
| 1 | 1 | | 1 | | 1 | 0 | | 0 | | 0 | 1 | | 1 |
| Mode-ABC | | 00 | | | 01 | | 11 | | | 10 | |
| 00 | | 0 | | | 0 | | 0 | | | 0 | |
| 01 | | 0 | | | 0 | | 0 | | | 0 | |
| 11 | | 0 | | | 1 | | 0 | | | 0 | |
| 10 | | 0 | | | 1 | | 0 | | | 0 | |

Kmap for out4-mode’AB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mode-ABC | 00 | 01 | 11 | 10 |
| 00 | 0 | 1 | 0 | 0 |
| 01 | 0 | 1 | 0 | 0 |
| 11 | 1 | 0 | 0 | 0 |
| 10 | 0 | 0 | 0 | 0 |

Kmap for out 3- mode’(ab’+a’bc)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mode-ABC | 00 | 01 | 11 | 10 |
| 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 1 | 0 | 0 |
| 11 | 0 | 1 | 0 | 0 |
| 10 | 1 | 0 | 0 | 0 |

Kmap for out2 =mode’a(b’+c)+mode’x’yz’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mode-ABC | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 0 | 0 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 0 | 0 | 1 | 1 |
| 10 | 1 | 1 | 1 | 0 |

Kmap for out 1=mode’(b xor c)+mode(ac+ab+bc)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Mode-ABC | 00 | 01 | 11 | 10 |
| 00 | 0 | 0 | 1 | 0 |
| 01 | 1 | 1 | 0 | 1 |
| 11 | 1 | 1 | 1 | 0 |
| 10 | 0 | 0 | 0 | 1 |

Kmap for out0=c(mode’+a xnor b)+mode c’(a xor b)

My code:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity kmapthree is

Port ( mode : in STD\_LOGIC;

a : in STD\_LOGIC;

b : in STD\_LOGIC;

c : in STD\_LOGIC;

output : out STD\_LOGIC\_vector(4 downto 0)); --have to extend cause 7 will go to 21 and 4 bits only covers up to 15

end kmapthree;

architecture Behavioral of kmapthree is

begin

output(0)<=(c and ((not mode) or (a xnor b))) or ((mode and (not c)) and (a xor b));

output(1)<=((not mode) and (b xor c)) or (Mode and ((a and c) or (a and b) or (b and c)));

output(2)<=(((not mode) and a) and ((not b) or c)) or ((not mode) and (not a) and b and (not c));

output(3)<=(not mode) and ((a and (not b)) or ((not a) and b and c));

output(4)<=(not mode) and a and b;

end Behavioral;

Simulations:

