# Designing Sum of Products (SOP)

## Introduction

Virtex-II slices contain a dedicated two-input multiplexer (MUXCY) and a two-input OR gate (ORCY) to perform operations involving wide AND and OR gates. These combine the four-input LUT outputs. These gates can be cascaded in a chain to provide the wide AND functionality across slices. The output from the cascaded AND gates can then be combined with the dedicated ORCY to produce the Sum of Products (SOP).

## Virtex-II CLB Resources

Each Virtex-II slice has a MUXCY, which uses the output from the LUTs as a SELECT signal. Depending on the width of data desired, several slices can be used to provide the SOP output. Figure 2-64 illustrates the logic involved in designing a 16-input AND gate. It utilizes the 4-input LUT to provide the necessary SELECT signal for the MUXCY. Only when all of the input signals are High, can the $V_{CC}$ at the bottom reach the output. This use of carry logic helps to perform AND functions at high speed and saves logic resources.

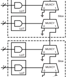

*Figure 2-64:* **Implementing a 16-bit Wide AND Gate Using MUXCY & ORCY**

The output from the chain of AND gates is passed as one of the inputs of the dedicated OR gate, ORCY. To calculate the SOP, these AND chains can be cascaded vertically across several CLBs, depending on the width of the input data. Figure 2-65 illustrates how the AND outputs are then passed in through the ORCY gates in a horizontal cascade, the sum of which is the Sum of Products.

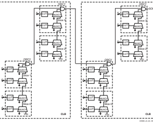

*Figure 2-65:* **64-bit Input SOP Design**

## Port Signals

### AND_WIDTH Parameter

The width of each AND gate used in the cascade.

### PROD_TERM Parameter

The number of AND gates used along each vertical cascade.

### AND_IN Parameter

Data input to the AND gates. The total width of data is calculated from the product of AND_WIDTH and PROD_TERM

### SOP_OUT Parameter

The Sum of Products (SOP) output data from the cascade chain.

## Applications

These logic gates can be used in various applications involving very wide AND gates and Sum of Products (SOP) functions.

## VHDL and Verilog Instantiation

To implement wide-input AND functions, MUXCY and ORCY primitives can be instantiated in VHDL or Verilog code. The submodule code provided can be used to implement wide-input AND gates for any width of input data.

### VHDL and Verilog Submodules

VHDL and Verilog submodules are available to implement the cascade chain of wide-input AND gates and OR gates to calculate the Sum of Products (SOP). The VHDL module provided uses a generic case, where the width of data and the product terms can be specified in the case. The Verilog module provides a 64-bit input example, using four wide AND chains, each of which handle 16 bits of data.

### VHDL Templates

```
-- Module : AND_CHAIN
-- Description : 16 input AND gate
--
-- Device : Virtex-II Family
---------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--library UNISIM;
--use UNISIM.VCOMPONENTS.ALL;

entity AND_CHAIN is
 generic (
    input_width : integer); --must be a 4x value
 port (
    data_in : in std_logic_vector( input_width-1 downto 0);
    carry_in : in std_logic;
    out_andor_chain : out std_logic);
end AND_CHAIN;

architecture AND_CHAIN_arch of AND_CHAIN is

component ORCY
  port( i : std_logic;
        ci : in std_logic;
        o : out std_logic);
end component;

component AND_LOGIC
  port( sel_data : in std_logic_vector(3 downto 0);
        data_cin : in std_logic;
        data_out : out std_logic);
end component;

signal VCC, GND : std_logic;
signal cout : std_logic_vector(input_width/4 downto 0);
signal out_and_chain : std_logic;

begin

VCC <= '1';
GND <= '0';

--initialization of first input for MUXCY
cout(0) <= VCC;

and_chain_x : for i in (input_width/4) - 1 downto 0 generate
```

```
  AND_LOGIC_inst : AND_LOGIC
        port map (
           sel_data => data_in((4 * i + 3) downto (4 * i)),
           data_cin => cout(i),
           data_out => cout(i + 1));
end generate;

out_and_chain <= cout(input_width/4);

orcy_inst : ORCY
  port map( i => out_and_chain,
        ci => carry_in,
        o => out_andor_chain);

end AND_CHAIN_arch;

---------------------------------------------------------------------
-- Module AND_LOGIC
-- Description : 4-input AND gate
--
-- Device : Virtex-II Family
---------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
--library UNISIM;
--use UNISIM.VCOMPONENTS.ALL;

entity AND_LOGIC is
  port(
      sel_data : in std_logic_vector(3 downto 0); -- data for select
signal for MUXCY from LUT
      data_cin : in std_logic;  -- result from previous stage
      data_out : out std_logic);
end AND_LOGIC;

architecture AND_LOGIC_arch of AND_LOGIC is

component MUXCY
 port(
     DI : in std_logic;
     CI : in std_logic;
     S : in std_logic;
     O : out std_logic);
end component;

signal GND : std_logic;
signal sel:std_logic;

begin

GND <= '0';
sel <= sel_data(0) and sel_data(1) and sel_data(2) and sel_data(3);

--Wide AND gate using MUXCY
MUX : MUXCY
     port map (
        DI => GND,
        CI => data_cin,
        S => sel,
        O => data_out);

end AND_LOGIC_arch;
```

```
-------------------------------------------------------------------
-- Module : SOP_SUBM
-- Description : Implementing SOP using MUXCY and ORCY
--
-- Device : Virtex-II Family
-------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
--library UNISIM;
--use UNISIM.VCOMPONENTS.ALL;

entity SOP_SUBM is
  generic(
     and_width : integer :=16 ;
     prod_term : integer := 4 );
  port(
     and_in : in std_logic_vector(and_width * prod_term - 1 downto 0);
     sop_out : out std_logic);
end SOP_SUBM;

architecture SOP_SUBM_arch of SOP_SUBM is

component AND_CHAIN
 generic (
    input_width : integer); --must be a 4x value
 port (
    data_in : in std_logic_vector( input_width-1 downto 0);
    carry_in : in std_logic;
    out_andor_chain : out std_logic);
end component;

signal VCC, GND : std_logic;
signal carry : std_logic_vector(prod_term downto 0);

begin

VCC <= '1';
GND <= '0';

carry(0) <= GND;
andor_inst : for i in 0 to (prod_term - 1) generate
      and_chainx : AND_CHAIN
             generic map(
                input_width => and_width)
             port map(
              data_in => and_in((and_width * i + (and_width -1)) downto
(and_width * i)),
                carry_in => carry(i),
                out_andor_chain => carry(i + 1));
end generate;
sop_out <= carry(prod_term);

end SOP_SUBM_arch;
```

Verilog Templates

```
// Module : AND_CHAIN
// Description : 16 input AND gate
//
// Device : Virtex-II Family
//---------------------------------------------------------------
module AND_CHAIN(data_in, carry_in, out_andor_chain);
input [15:0] data_in;
input carry_in;
output out_andor_chain;

wire VCC = 1'b1;
wire out_and_chain;
wire dat_out1, data_out2, data_out3;

AND_LOGIC u4(.sel_data(data_in[15:11]), .data_cin(data_out3),
.data_out(out_and_chain));

AND_LOGIC u3(.sel_data(data_in[10:8]), .data_cin(data_out2),
.data_out(data_out3));

AND_LOGIC u2(.sel_data(data_in[7:4]), .data_cin(data_out1),
.data_out(data_out2));

AND_LOGIC u1(.sel_data(data_in[3:0]), .data_cin(VCC),
.data_out(data_out1));

orcy  u5(.i(out_and_chain), .ci(carry_in), .o(out_andor_chain));

endmodule
//---------------------------------------------------------------
// Module AND_LOGIC
// Description : 4-input AND gate
//
// Device : Virtex-II Family
//---------------------------------------------------------------
// Module : init_and
//
module AND_LOGIC(sel_data, data_cin, data_out);

input[3:0] sel_data;
input data_cin;

output data_out;

wire GND = 1'b0;
wire VCC = 1'b1;
wire and_out;

assign and_out = sel_data[3] & sel_data[2] & sel_data[1] & sel_data[0];

MUXCY  muxcy_inst (.DI(GND), .CI(data_cin), .s(and_out), .o(data_out));

endmodule
//---------------------------------------------------------------
// Module : SOP_SUBM
// Description : Implementing SOP using MUXCY and ORCY
//
// Device : Virtex-II Family
//---------------------------------------------------------------
`include "AND_LOGIC.v"
```

```
`include "AND_CHAIN.v"


module SOP_SUBM(and_in, sop_out);

input [63:0] and_in;
output sop_out;
wire out_andor_chain1, out_andor_chain2, out_andor_chain3;
wire GND = 1'b0;

AND_CHAIN u4(.data_in(and_in[63:48]), .carry_in(out_andor_chain3),
               .out_andor_chain(sop_out));

AND_CHAIN u3(.data_in(and_in[47:32]), .carry_in(out_andor_chain2),
               .out_andor_chain(out_andor_chain3));

AND_CHAIN u2(.data_in(and_in[31:16]), .carry_in(out_andor_chain1),
              .out_andor_chain(out_andor_chain2));

AND_CHAIN u1(.data_in(and_in[15:0]), .carry_in(GND),
               .out_andor_chain(out_andor_chain1));

endmodule
```