Advanced Digital Circuit Design - HDL Models of Combinational Circuits

Prof. Dr. Berna Örs Yalçın

Istanbul Technical University

Faculty of Electrical and Electronics Engineering

Department of Electronics and Communication Engineering

siddika.ors@itu.edu.tr

The Three Approaches for Describing a Digital Circuit in Verilog

1. Gate-Level Modeling
2. Dataflow Modeling
3. Behavioral Modeling

1. Gate-Level (Structural) Modeling

- Digital circuit is described by using only instantiations of predefined primitives (primitive gates), user-defined primitives and user defined modules.

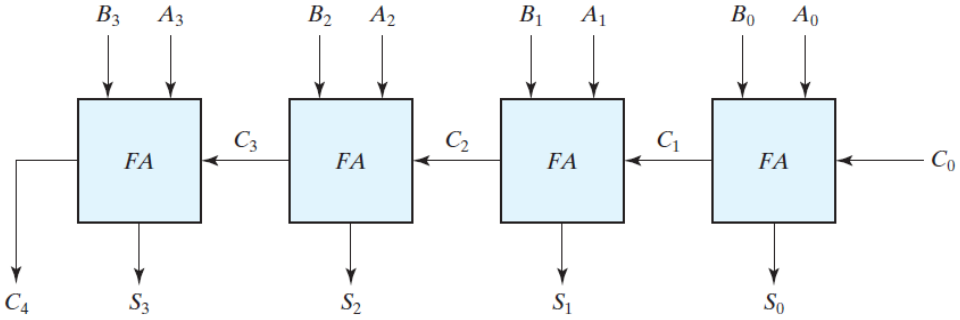
- Describes a circuit by specifying its logic gates and their interconnections; provides a textual desription of a schematic diagram.

- Verilog has 12 basic gates as predefined primitives. 4 of these are of three-state type, meaning they may take a high-impedance (z) output value.

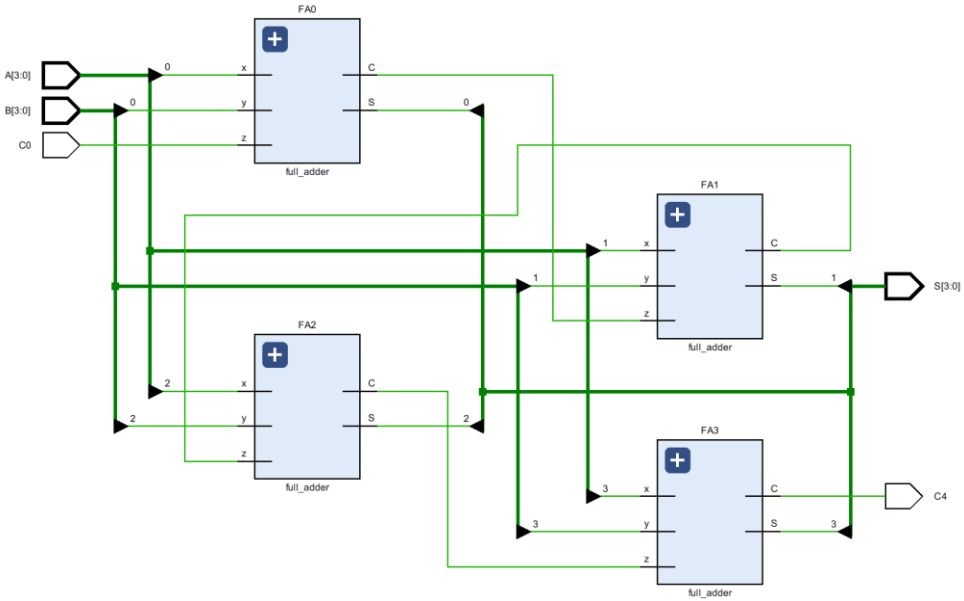
  and, **nand**, **or**, **nor**, **xor**, **xnor** : n-input primitives
  not, **buf** : n-output primitives

- **Logic values:** 0, 1, x (don't care / unknown), z (high-impedance)

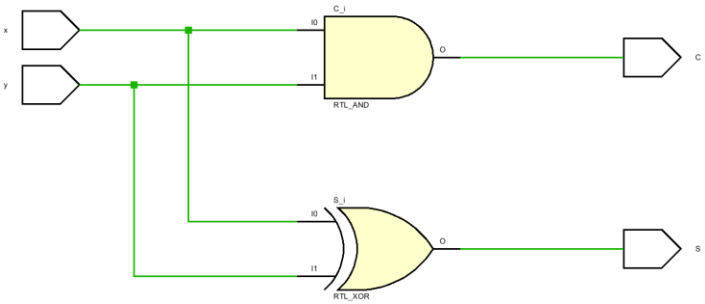
Gate-Level VHDL Description of a 4-bit Adder



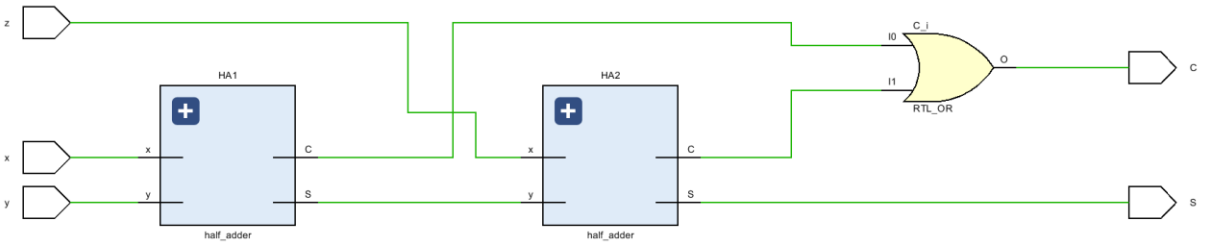

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ripple_carry_adder is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           C0 : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (3 downto 0);
           C4 : out STD_LOGIC);
end ripple_carry_adder;

architecture Gate_Level of ripple_carry_adder is
    component full_adder is
        Port ( x : in STD_LOGIC;
               y : in STD_LOGIC;
               z : in STD_LOGIC;
               S : out STD_LOGIC;
               C : out STD_LOGIC);
    end component;
    signal C : STD_LOGIC_VECTOR (3 downto 1);
begin
    FA0: full_adder Port map(A(0),B(0),C0,S(0),C(1));
    FA1: full_adder Port map(A(1),B(1),C(1),S(1),C(2));
    FA2: full_adder Port map(A(2),B(2),C(2),S(2),C(3));
    FA3: full_adder Port map(A(3),B(3),C(3),S(3),C4);
end Gate_Level;
```



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity half_adder is
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           S : out STD_LOGIC;
           C : out STD_LOGIC);
end half_adder;
architecture Gate_Level of half_adder is
begin
    S <= x xor y;
    C <= x and y;
end Gate_Level;
```

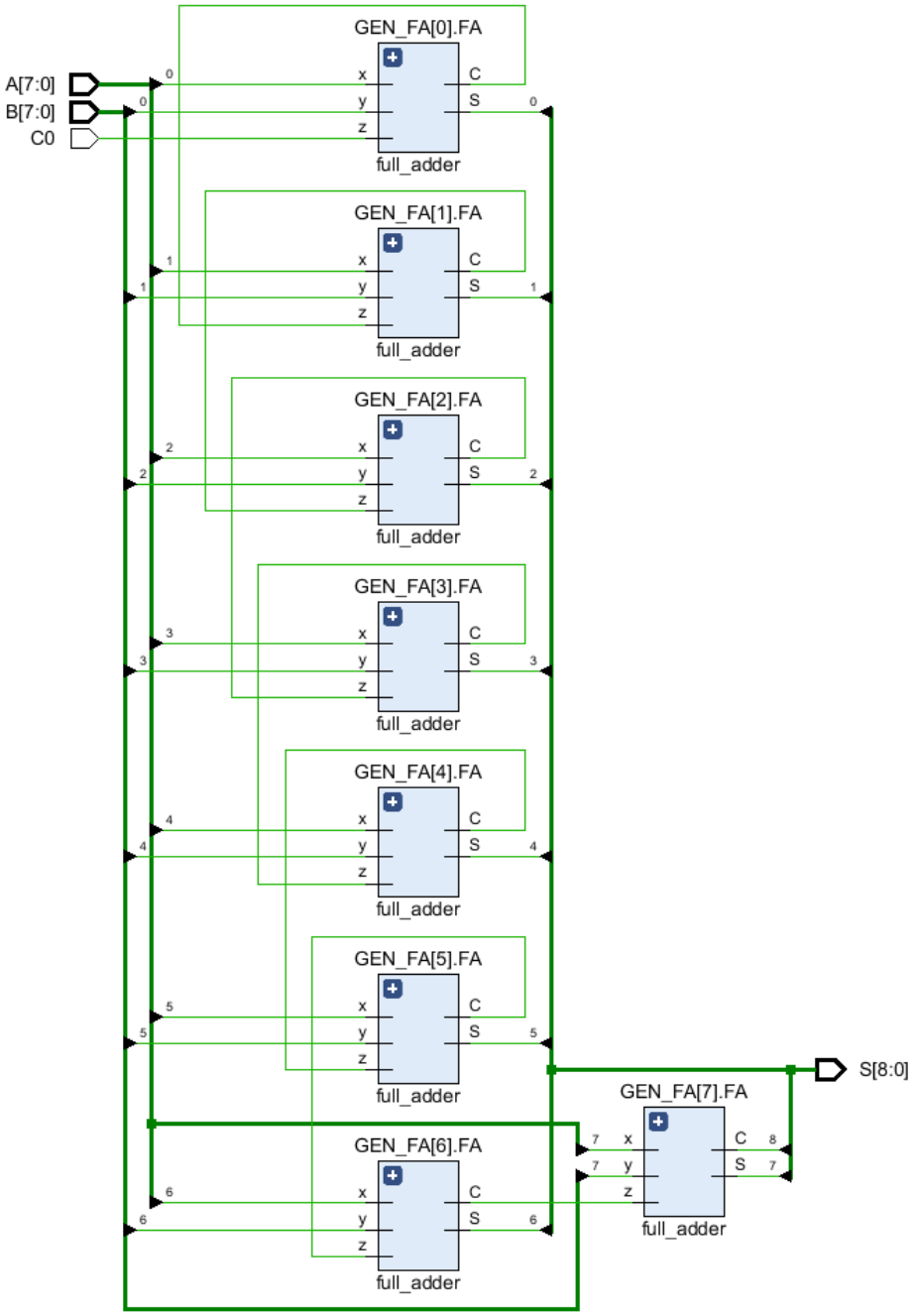



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity full_adder is
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           z : in STD_LOGIC;
           S : out STD_LOGIC;
           C : out STD_LOGIC);
end full_adder;
architecture Gate_Level of full_adder is
    component half_adder is
        Port ( x : in STD_LOGIC;
               y : in STD_LOGIC;
               S : out STD_LOGIC;
               C : out STD_LOGIC);
    end component;
    signal S1,C1,C2 : std_logic;
begin
    HA1: half_adder Port map(x => x,
                             y => y,
                             S => S1,
                             C => C1);
    HA2: half_adder Port map(x => z,
                             y => S1,
                             S => S,
                             C => C2);
    C <= C1 or C2;
end Gate_Level;
```

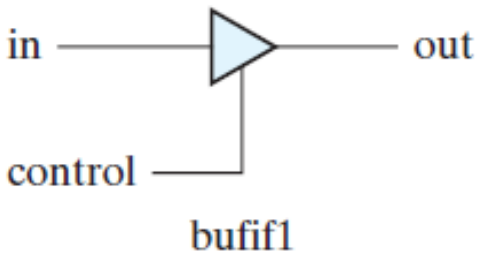


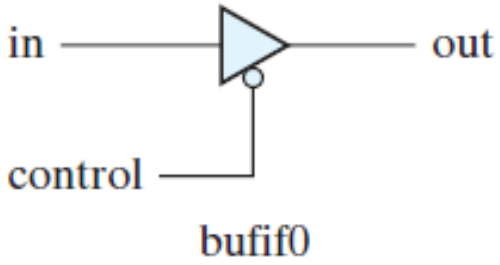
Gate-Level VHDL Description of a n-bit Adder

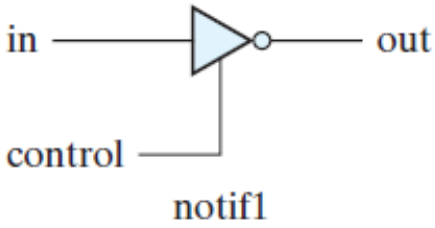
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity ripple_carry_adder is
    generic (n : integer := 8);
    Port ( A : in STD_LOGIC_VECTOR (n-1 downto 0);
           B : in STD_LOGIC_VECTOR (n-1 downto 0);
           C0 : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (n downto 0));
end ripple_carry_adder;

architecture Gate_Level of ripple_carry_adder is
    component full_adder is
        Port ( x : in STD_LOGIC;
               y : in STD_LOGIC;
               z : in STD_LOGIC;
               S : out STD_LOGIC;
               C : out STD_LOGIC);
    end component;
    signal C : STD_LOGIC_VECTOR (n downto 0);
begin
    C(0) <= C0;
    GEN_FA: for i in 0 to n-1 generate
        FA: full_adder Port map(A(i),B(i),C(i),S(i),C(i+1));
    end generate GEN_FA;
    S(n) <= C(n);
end Gate_Level;
```

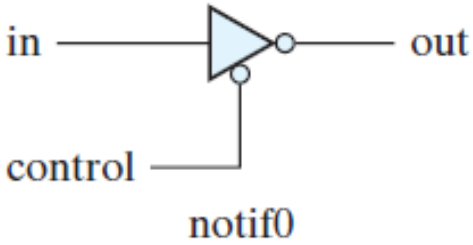


Three-State Gates



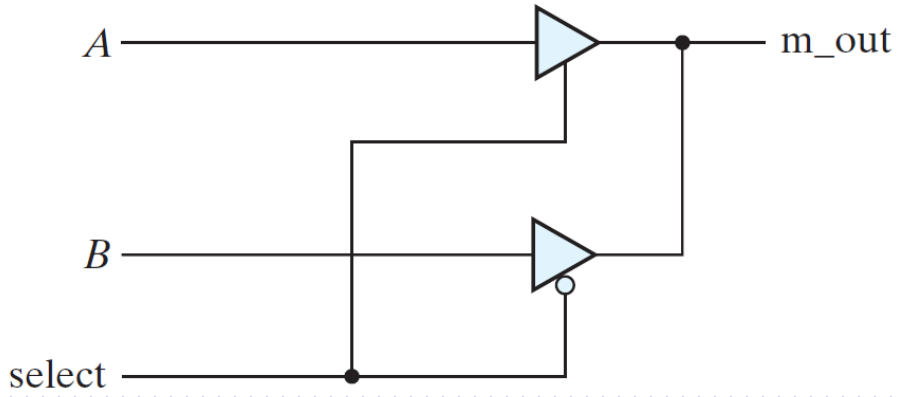




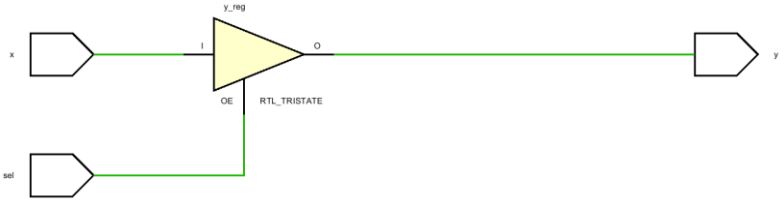


- A three-state gate has a control input that can place the gate into the high-impedance state (denoted by z).
- Outputs of three-state gates can have multiple drivers. In Verilog, the keyword **tri** is used to define an output of **tri** data type and to indicate that an output has multiple drivers.

Gate-Level VHDL Description of a 2x1 Multiplexer with aThree-State Output

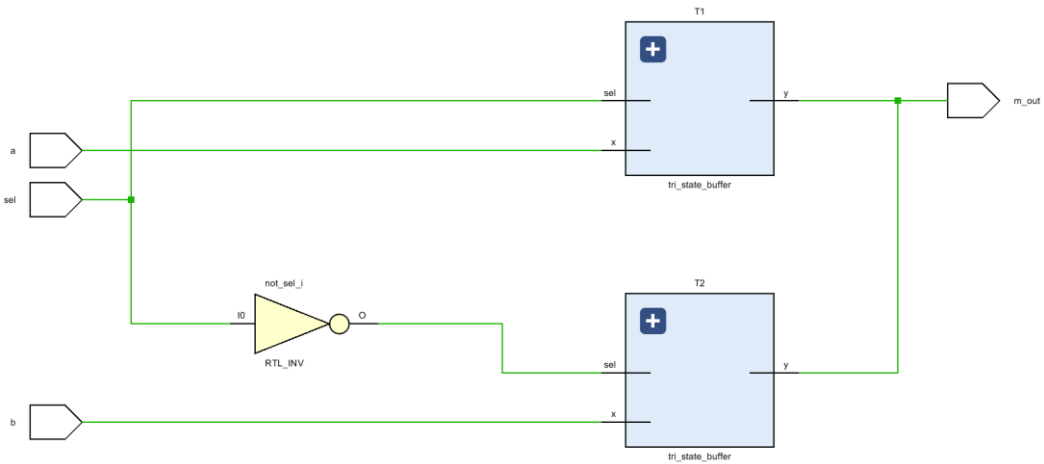



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tri_state_buffer is
    Port ( x : in STD_LOGIC;
           sel : in STD_LOGIC;
           y : out STD_LOGIC);
end tri_state_buffer;
architecture Behavioral of tri_state_buffer is
begin
    process(x,sel)
    begin
        if(sel='1') then
            y <= x;
        else
            y <= 'Z';
        end if;
    end process;
end Behavioral;
```



Gate-Level VHDL Description of a 2x1 Multiplexer with aThree-State Output

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux_tri is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           sel : in STD_LOGIC;
           m_out : out STD_LOGIC);
end mux_tri;

architecture Gate_Level of mux_tri is
    component tri_state_buffer is
        Port ( x : in STD_LOGIC;
               sel : in STD_LOGIC;
               y : out STD_LOGIC);
    end component;
    signal not_sel : std_logic;
begin
    not_sel <= not sel;
    T1: tri_state_buffer Port map(a,sel,m_out);
    T2: tri_state_buffer Port map(b,not_sel,m_out);
end Gate_Level;
```



2. Dataflow Modeling

- Data-flow HDL models describe combinational circuits by their function rather than by their gate structure.
- In data-flow modeling, a number of operators are used on operands to produce the desired result.
- Uses continuous assignments with the keyword '<=' .
- A continuous assignment assigns a value to a net (declared explicitly by the keyword **signal** or implicitly as an output port) which represents a physical connection between circuit elements.

Example:

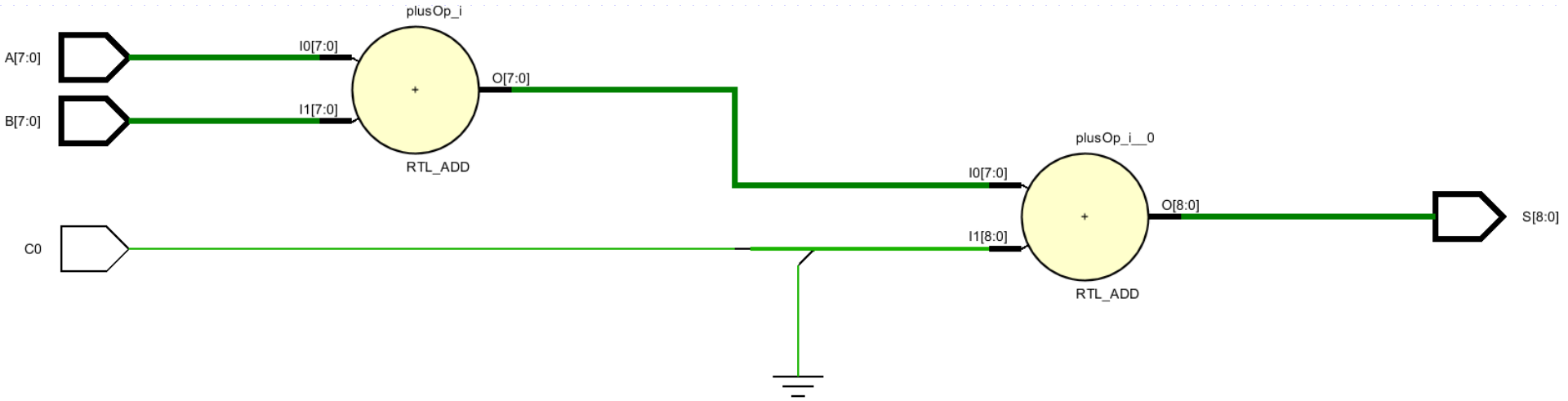
```
Y <= (A and S) or (B and not S);
```

VHDL Operators

**	exponentiation,	numeric ** integer,	result numeric
abs	absolute value,	abs numeric,	result numeric
not	complement,	not logic or boolean,	result same
*	multiplication,	numeric * numeric,	result numeric
/	division,	numeric / numeric,	result numeric
mod	modulo,	integer mod integer,	result integer
rem	remainder,	integer rem integer,	result integer
+	addition,	numeric + numeric,	result numeric
-	subtraction,	numeric - numeric,	result numeric
&	concatenation,	array or element & array or element,	result array
sll	shift left logical,	logical array sll integer,	result same
srl	shift right logical,	logical array srl integer,	result same
sla	shift left arithmetic,	logical array sla integer,	result same
sra	shift right arithmetic,	logical array sra integer,	result same
rol	rotate left,	logical array rol integer,	result same
ror	rotate right,	logical array ror integer,	result same
=	test for equality,		result is boolean
/=	test for inequality,		result is boolean
<	test for less than,		result is boolean
<=	test for less than or equal,		result is boolean
>	test for greater than,		result is boolean
>=	test for greater than or equal,		result is boolean
and	logical and,	logical array or boolean,	result is same
or	logical or,	logical array or boolean,	result is same
nand	logical complement of and,	logical array or boolean,	result is same
nor	logical complement of or,	logical array or boolean,	result is same
xor	logical exclusive or,	logical array or boolean,	result is same
xnor	logical complement of exclusive or,	logical array or boolean,	result is same

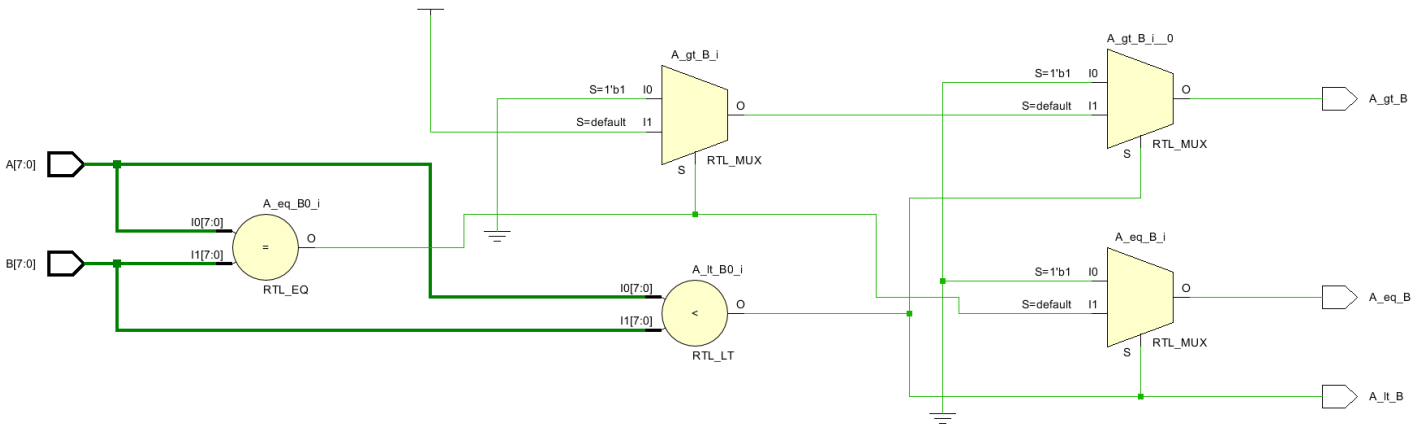
Dataflow VHDL Description of a 4-Bit Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ripple_carry_adder is
    generic (n : integer := 8);
    Port ( A : in STD_LOGIC_VECTOR (n-1 downto 0);
           B : in STD_LOGIC_VECTOR (n-1 downto 0);
           C0 : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (n downto 0));
end ripple_carry_adder;

architecture Dataflow of ripple_carry_adder is
    signal C : STD_LOGIC_VECTOR (n downto 0);
begin
    C(0) <= '0';
    C(n downto 1) <= (others => '0');
    S <= A + B + C;
end Dataflow;
```



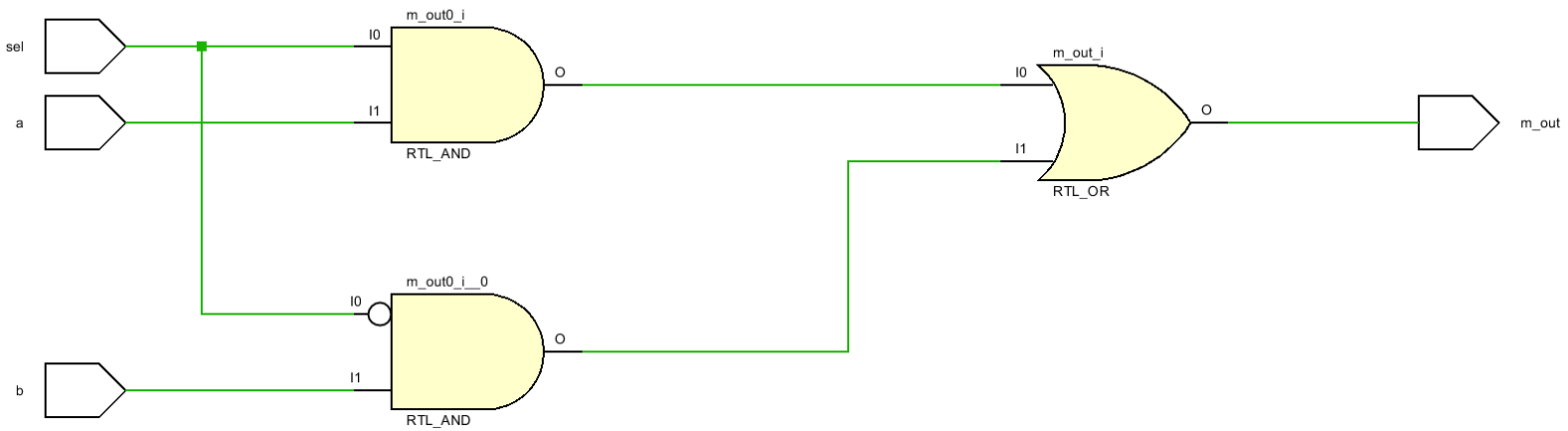
Dataflow VHDL Description of a n-Bit Comparator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity compare is
    generic (n: integer :=8);
    Port ( A : in STD_LOGIC_VECTOR (n-1 downto 0);
           B : in STD_LOGIC_VECTOR (n-1 downto 0);
           A_lt_B : out STD_LOGIC;
           A_eq_B : out STD_LOGIC;
           A_gt_B : out STD_LOGIC);
end compare;

architecture Behavioral of compare is
begin
    process(A,B)
    begin
        if(A<B) then
            A_lt_B <= '1';
            A_eq_B <= '0';
            A_gt_B <= '0';
        elsif(A=B) then
            A_lt_B <= '0';
            A_eq_B <= '1';
            A_gt_B <= '0';
        else
            A_lt_B <= '0';
            A_eq_B <= '0';
            A_gt_B <= '1';
        end if;
    end process;
end Behavioral;
```



Dataflow VHDL Description of a 2x1 Multiplexer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux_tri is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           sel : in STD_LOGIC;
           m_out : out STD_LOGIC);
end mux_tri;

architecture Dataflow of mux_tri is
begin
    m_out <= (sel and a) or (not sel and b);
end Dataflow;
```

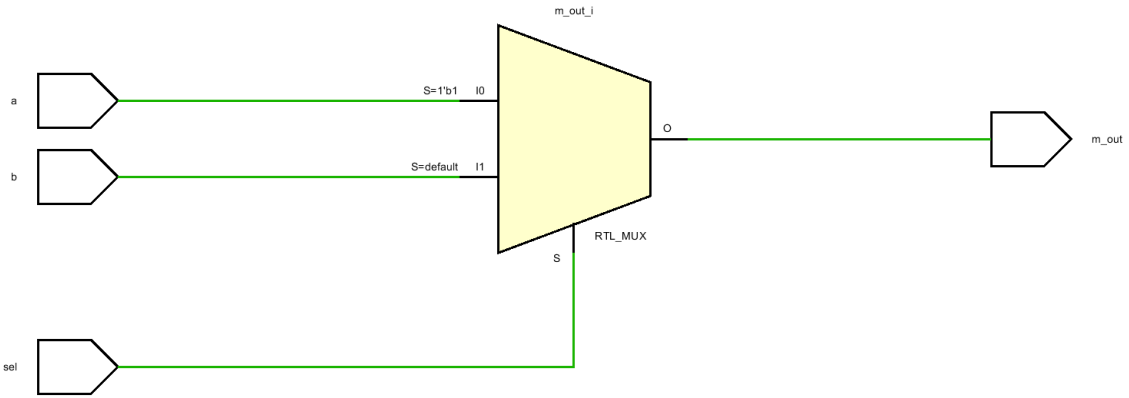


3. Behavioral Modeling

- Used to describe circuits at a higher level of abstraction, at a functional and algorithmic level.
- Used mostly to describe sequential circuits, but can be used to describe combinational circuits as well.
- Uses the keyword **process**, followed by an optional event control expression and a list of procedural assignment statements.
- An event control expression specifies when the statements will execute.

Behavioral VHDL Description of a 2x1 Multiplexer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux_tri is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           sel : in STD_LOGIC;
           m_out : out STD_LOGIC);
end mux_tri;

architecture Behavioral of mux_tri is
begin
    process(a,b,sel)
    begin
        if(sel='1') then
            m_out <= a;
        else
            m_out <= b;
        end if;
    end process;
end Behavioral;
```



Behavioral VHDL Description of 4x1 Multiplexer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux_4bit is
    Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
           sel : in STD_LOGIC_VECTOR (1 downto 0);
           m_out : out STD_LOGIC);
end mux_4bit;

architecture Behavioral of mux_4bit is
begin
    process(a,sel)
    begin
        case sel is
            when "00" => m_out <= a(0);
            when "01" => m_out <= a(1);
            when "10" => m_out <= a(2);
            when "11" => m_out <= a(3);
        end case;
    end process;
end Behavioral;
```