COMBINATIONAL CIRCUITS

A combinational circuit is a type of digital logic circuit in which the output is solely determined by the current inputs, without any memory or feedback loops. It consists of logic gates that combine the input signals to produce an output based on Boolean algebra.

|  |  |
| --- | --- |
| ADVANTAGES | DISADVANTAGE |
| Simple to design and implement. | More prone to glitches. |
| Faster operation due to lack of memory elements. | Complexity increases with the number of inputs. |
| Lower power consumption compared to sequential circuits. | Static functionality; limited to current input. |
| Predictable behavior, making testing easier. | Cannot store or process sequential data. |
| Useful for arithmetic, data selection, and encoding/decoding. | High hardware requirements for complex designs. |

IMPORTANT

LOGICAL EQUATIONS – Concise, Clarity and Optimized synthesis but limited Flexibility and scalability also No explicit Control flow.

IF ELSE – Synthesize to nested 2:1 MUX with PERIORITY ENCODING.

CASE STATEMENT – Synthesize to MUX OF 2n :1.





MUTIPLEXER

It receives binary information from several input lines (2n ) and route it to a single output according to select line( N)

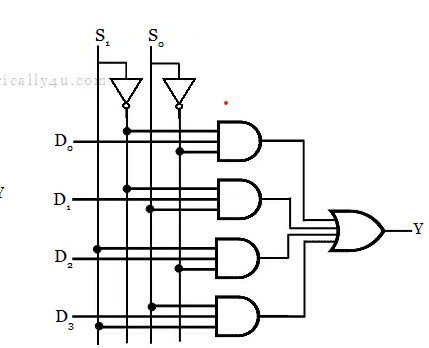
2n:1.

Verilog Implementations of 4:1

DEMUTIPLEXER

It is digital circuit that takes a single input signal and directs it to one of many output lines based on the values of select lines.

**USING LOGICAL :**



assign Y0 = ~A2 & ~A1 & ~A0;

assign Y1 = ~A2 & ~A1 & A0;

assign Y2 = ~A2 & A1 & ~A0;

assign Y3 = ~A2 & A1 & A0;

assign Y4 = A2 & ~A1 & ~A0;

assign Y5 = A2 & ~A1 & A0;

assign Y6 = A2 & A1 & ~A0;

assign Y7 = A2 & A1 & A0;

ENCODER

An **encoder** is a combinational circuit that converts a set of **2n**inputs into an **n-bit** output, where only one input is high (1) at any time.

**Using Conditional(Ternary) Operator-**

assign Y = S1 ? (S0 ? I3 : I2)

: (S0 ? I1 : I0);

case ({I3, I2, I1, I0})

4'b0001: {Y1, Y0} = 2'b00;

4'b0010: {Y1, Y0} = 2'b01;

4'b0100: {Y1, Y0} = 2'b10;

4'b1000: {Y1, Y0} = 2'b11;

default: {Y1, Y0} = 2'b00;

endcase

**Using if-else statements [PERIORITY]**

if (sel == 2'b00) out = data[0];

else if (sel == 2'b01) out = data[1]; else if (sel == 2'b10) out = data[2];

else if (sel == 2'b11) out = data[3];

else out = 1'b00; // Default

**Using case statement –**

case (sel)

2'b00: out = data [0]; // Select input 0 2'b01: out = data [1]; // Select input 1 2'b10: out = data [2]; // Select input 2 2'b11: out = data [3]; // Select input 3 default: out = 1'b0; // avoid latch

endcase

DECODER

A **decoder** converts an **n-bit input** into a one-hot output, where each input combination produces a unique output.

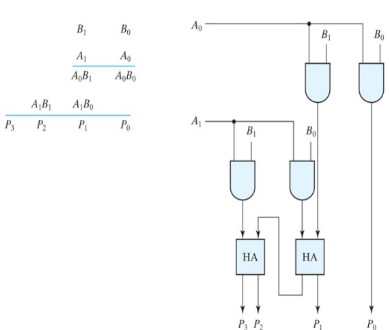
assign Y0 = ~A1 & ~A0; // A1' A0' assign Y1 = ~A1 & A0; // A1' A0 assign Y2 = A1 & ~A0; // A1 A0' assign Y3 = A1 & A0; // A1 A0

MULTIPLIER

. A **Binary Multiplier** is a digital circuit used to multiply two binary numbers.

#P0=A0.B0 # P1=A1B0+A0.B1

#P2=A1B1+A0B1 #P3=A1B1



module binary\_multiplier (

input [1:0] A ,input [1:0] B,

output [3:0] P );

assign P[0] = A[0] & B[0];

assign P[1] = (A[1] & B[0]) ^ (A[0] & B[1]);

assign P[2] = (A[1] & B[1]) ^ ((A[0] & B[0]) & (A[1] & B[0]));

assign P[3] = A[1] & B[1];

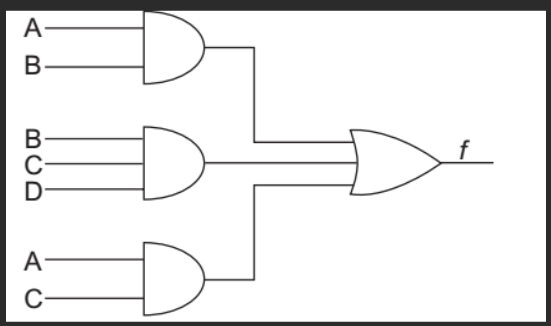
endmodule

MAJORITY VOTER

A **Majority Voter** is a digital circuit that outputs the value that occurs most frequently among its inputs.

**Majority = 1** if at least two of the three inputs are 1.

**Majority = 0** if at least two of the three inputs are 0.



module majority\_voter( input A, // First input bit

input B, // Second input bit

input C, // Third input bit

output Majority // Majority output (0 or 1)

);

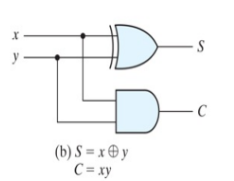
// Majority logic: Majority = (A & B) | (B & C) | (A & C)

assign Majority = (A & B) | (B & C) | (A & C);

endmodule

HALF ADDER

add two single-bit binary numbers. It has two inputs (A and B) and two outputs: the **Sum** and the **Carry**.



module half\_adder(

input A, // First input bit

input B, // Second input bit

output Sum, // Sum output

output Carry // Carry output

);

// Sum is the XOR of A and B

assign Sum = A ^ B;

// Carry is the AND of A and B

assign Carry = A & B;

endmodule

A B Bin| D Bout

0 0 0 | 0 0

0 0 1 | 1 1

0 1 0 | 1 1

0 1 1 | 0 1

1 0 0 | 1 0

1 0 1 | 0 1

1 1 0 | 0 0

1 1 1 | 1 1

X Y C S

0 0 0 0

0 1 0 1

1 0 0 1

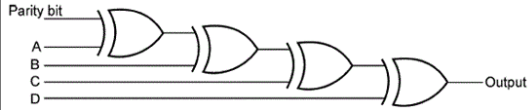
1 1 1 0

PARITY GENERATOR

A **Parity Detector** is a digital circuit that checks whether a given set of bits has **even** or **odd parity**. Parity refers to the number of 1 bits in a binary number:

**Even parity**: The number of 1 bits is even.

**Odd parity**: The number of 1 bits is odd.



module parity\_detector (

input [3:0] A, // 4-bit input A

output parityeven // Even parity

output parityodd //odd parity

);

// Even parity is the XOR of all input bits

assign parityeven = A[0] ^ A[1] ^ A[2] ^ A[3];

assign parityodd = ~(A[0] ^ A[1] ^ A[2] ^ A[3]);

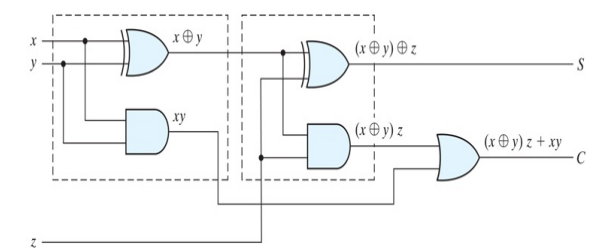
endmodule

FULL ADDER

A Full Adder is a combinational circuit that adds three bits: two input bits and a carry-in (from a previous stage). It produces two outputs: the Sum and the Carry-out.

**Sum (S)** = A ⊕ B ⊕ Cin =∑(1,2,4,7)

**Carry-out (Cout)** = AB+BC+AC



module full\_adder(

input A, input B , input Cin

output Sum, output Cout

);

// Sum is the XOR of A, B, and Cin

assign Sum = A ^ B ^ Cin;

// Carry-out is the OR of the AND combinations of A, B, and Cin

assign Cout = (A & B) | (B & Cin) | (A & Cin);

endmodule

FULL SUBTRACTOR

**Full Subtractor** is a combinational circuit that subtracts three binary bits: two input bits (A and B) and a borrow-in (Bin from a previous stage or 0). It produces two outputs: the **difference (D)** and the **borrow-out (Bout)**.

**Difference (D)** = A ⊕ B ⊕ Bin

**Borrow-out (Bout)** = ~AB +BBin+ ~AB

module full\_subtractor(

input A, input B , input Bin ,output D,OUTPUT Bout );

assign D = A ^ B ^ Bin;

assign Bout = (~A & B) | (~A & Bin) | (B & Bin);

endmodule

**DESIGN PROCEDURE**

>>From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.

>>Derive the truth table that defines the required relationship between

inputs and outputs.

>>Obtain the simplified Boolean functions for each output as a function of the input variables(K-MAP).

>>Draw the logic diagram and verify it.

correctness of the design.

**MUlTIPY CONSTANT**

Let, 3A =2A+A;

Hence, shift once and then add with number

module MutiPY (

input [3:0]NUM, input [3:0]multiplier,

output reg [7:0] result );

integer i;

always @(\*) begin

result = 8'd0;

for (i = 0; i < 4; i = i + 1) begin

if (multiplier[i] == 1'b1) begin

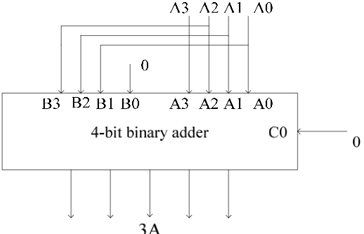
result = result + (NUM<< i);

end

end

end

endmodule



**Carry Look Ahead Adder**

To overcome ripple carry delay, we can anticipate carry bit generation in advance. By analyzing input bits early, we can determine if a carry will occur, reducing delay.

Pi = Ai ⊕ Bi  Gi = Ai Bi



module CarryLookAheadAdder(

input [3:0]A, B, input Cin,

output [3:0] S,output Cout);

wire [3:0] Ci;

assign Ci[0] = Cin;

assign Ci[1] = (A[0] & B[0]) | ((A[0]^B[0])

& Ci[0]);

assign Ci[2] = (A[1] & B[1]) | ((A[1]^B[1]) & ((A[0] & B[0]) | ((A[0]^B[0]) & Ci[0]))); assign Ci[3] = (A[2] & B[2]) | ((A[2]^B[2]) & ((A[1] & B[1]) | ((A[1]^B[1]) & ((A[0] & B[0]) | ((A[0]^B[0]) & Ci[0])))));

assign Cout = (A[3] & B[3]) | ((A[3]^B[3]) & ((A[2] & B[2]) | ((A[2]^B[2]) & ((A[1] & B[1]) | ((A[1]^B[1]) & ((A[0] & B[0]) | ((A[0]^B[0]) & Ci[0])))))));

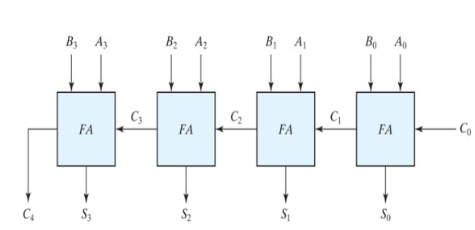
assign S = A^B^Ci;

endmodule

**Ripple Carry Adder**

A structure of multiple full adders is cascaded in a manner to gives the results of the addition of an n bit binary sequence.

Tdelay=(n-1)Tcarry +max(Tcarry,Tsum)



module ripple\_carry\_adder(a, b, cin, sum, cout);  
input [3:0] a; input [3:0] b; input cin; output [3:0] sum; output cout;  
wire [2:0]c;  
fulladd a1(a[0],b[0],cin, sum[0],c[0]);  
fulladd a2(a[1],b[1],c[0],sum[1],c[1]);  
fulladd a3(a[2],b[2],c[1],sum[2],c[2]);  
fulladd a4(a[3],b[3],c[2],sum[3],cout);  
endmodule

\*\*full\_adder module is in full adder

Multiply by even number :

Shift left using << operator.

Divide by even number :

Shift right using >> operator.

Check if to numbers are equal:

XNOR the numbers.

Check if to numbers are not equal:

XOR the numbers.

Digital circuits negative numbers are represented by using 2’s complement .

**ABSOLUTE OF VALUE**



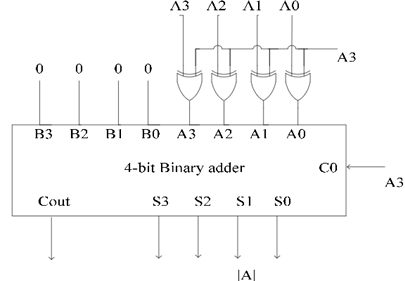
module Absolute\_Value (

input [3:0] value,

output [3:0] abs\_value );

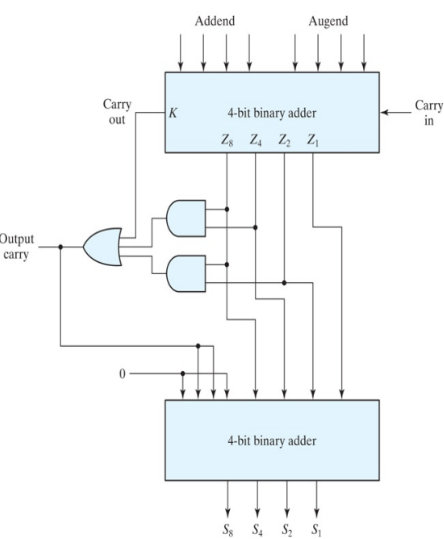
assign abs\_value = (value[3] == 1'b1) ? (~value + 4'b0001) : value;

*// Take 2's complement if negative*

endmodule

**BCD Adder**

BCD number can represent 0000-1001,so when the sum to two BCD number greater then 9 ,need to add 0110 to the resulted sum to make it vaild. Use full\_adder module.



module BCD\_Adder (

input [3:0] a, *// 4-bit BCD input A*

input [3:0] b, // 4-bit BCD input B

output [3:0] sum*, // 4-bit BCD sum*

output carry // *Carry out*

);

wire [4:0] temp\_sum; // *Temporary 5-bit* sum to handle carry

assign temp\_sum = a + b; *// Add the BCD numbers*

assign carry = (temp\_sum > 4'd9*); // Carry occurs if sum exceeds 9*

assign sum = carry ? (temp\_sum + 4'd6) : temp\_sum; *// Adjust for BCD if carry*

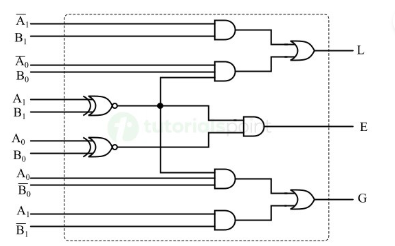
endmodule

**MAGINTUDE COMPARATOR**

A=B: E=(A0⊙B0)(A1⊙B1)

A<B: L=~A1B1+(A1⊙B1)(~A0)B0

A>B: G=A1(~B1)+(A1⊙B1)A0(~B0)



module magComp (  
input [7:0] In1,input [7:0] In2,  
output Gt,output Lt,output Eq);

reg Gt, Lt, Eq;

always @ (\*)  
begin  
 Gt <= (In1 > In2) ? 1’b1 : 1’b0;  
 Lt <= (In1 < In2) ? 1’b1 : 1’b0;  
 Eq <= (In1 == In2) ? 1’b1 : 1’b0;  
 end

endmodule

R

* IIT KHARAGPUR **: PROF. INDRANIL SENGUPTA** ( HARDWARE MODELING USING VERILOG)

<https://www.youtube.com/watch?v=NCrlyaXMAn8&list=PLJ5C_6qdAvBELELTSPgzYkQg3HgclQh-5>

* **ANKIT GOYAL SIR**: DIGITAL ELECTRONICS

<https://youtube.com/playlist?list=PLs5_Rtf2P2r41iuDKULDHHnIwfXyTAxBH&si=PhMhOvo8_rT1a53y>

* BOOKS:

DIGITAL ELECTRONICS: **Digital Logic and Computer Design by M. Morris Mano.**

VERILOG : **Verilog HDL - Samir Palnitkar**

Digital Design With an Introduction to the Verilog HDL, VHDL, and SystemVerilog by **M. Morris Mano and Michael D. Ciletti**