****

****

**Project Report**

**Topic: 16-Bit Barrel Shifter**

**Group members:**

Muhammad Usman (cs211208)

Saad Nisar Butt (cs211246)

**Literature Review:**

A barrel shifter is a digital circuit that can shift a data word by a specified number of bits without the use of any sequential logic, only pure combinational logic. It has a control input that specifies the number of bit positions that it shifts by. The Barrel Shifter is similar to the Shift Register (Multi-bit), except that bits shifted of the register are shifted back into the opposite end of the register. It rotates the bit to be shifted. For example, in right shift operations, the LSBs shifted out of the register are shifted into the MSBs.

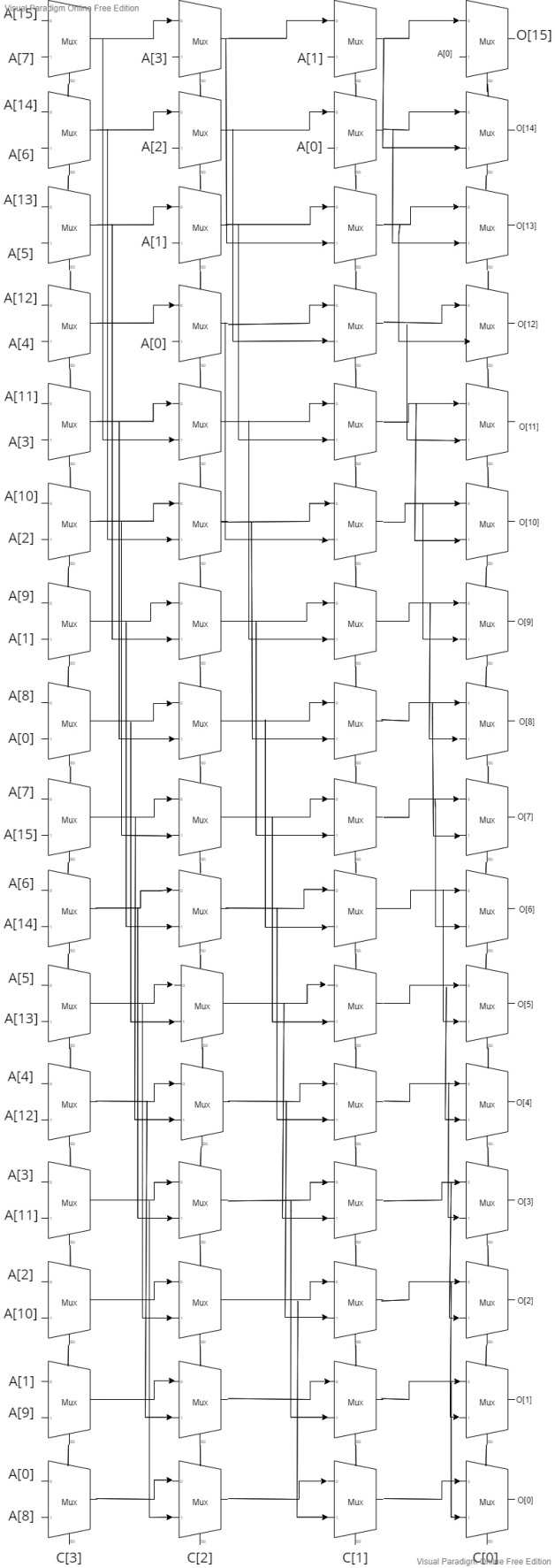
It can be implemented as a sequence of multiplexers. In this implementation, the output of one MUX is connected to the input of the next MUX in a way that depends on the shift distance. The number of multiplexers required is **n\*log2(n)** , for an **n** bit word.

For 16-bit barrel shifter --- 16\*log2(16) = 64

i.e. 64 multiplexers are required to implement the circuitry of 16-bit barrel shifter.

For the right shift operation, a 2x1 mux is used as a module. A control signal will determine the number of bits to be shifted. When implementing a barrel shifter with a sequence of shift multiplexers, each shifts a word by 2^k bit positions (1,2,4,8...) for different values of k. The number of multiplexing stages is relative to the width of the input. For 16-bit input, there are 4 stages. Each stage’s shifting depends on control signal’s bit.

**Block Diagram:**

****

**Boolean Equation:**

**For 2x1 MUX**

**Verilog Code:**

**design.v**

`include "mux.v"

module barrel(A,C,O);

    input[15:0] A;

    input[3:0] C;

    output[15:0] O;

    wire [15:0] x,y,z;

    //8bit shift right

    mux mux\_15(A[15], A[7], C[3], x[15]);

    mux mux\_14(A[14], A[6], C[3], x[14]);

    mux mux\_13(A[13], A[5], C[3], x[13]);

    mux mux\_12(A[12], A[4], C[3], x[12]);

    mux mux\_11(A[11], A[3], C[3], x[11]);

    mux mux\_10(A[10], A[2], C[3], x[10]);

    mux mux\_9(A[9], A[1], C[3], x[9]);

    mux mux\_8(A[8], A[0], C[3], x[8]);

    mux mux\_7(A[7], A[15], C[3], x[7]);

    mux mux\_6(A[6], A[14], C[3], x[6]);

    mux mux\_5(A[5], A[13], C[3], x[5]);

    mux mux\_4(A[4] ,A[12], C[3], x[4]);

    mux mux\_3(A[3], A[11], C[3], x[3]);

    mux mux\_2(A[2], A[10], C[3], x[2]);

    mux mux\_1(A[1], A[9], C[3], x[1]);

    mux mux\_0(A[0], A[8], C[3], x[0]);

    //4bit shift right

    mux mux\_31(x[15], A[3], C[2], y[15]);

    mux mux\_30(x[14], A[2], C[2], y[14]);

    mux mux\_29(x[13], A[1], C[2], y[13]);

    mux mux\_28(x[12], A[0], C[2], y[12]);

    mux mux\_27(x[11], x[15], C[2], y[11]);

    mux mux\_26(x[10], x[14], C[2], y[10]);

    mux mux\_25(x[9], x[13], C[2], y[9]);

    mux mux\_24(x[8], x[12], C[2], y[8]);

    mux mux\_23(x[7], x[11], C[2], y[7]);

    mux mux\_22(x[6], x[10], C[2], y[6]);

    mux mux\_21(x[5], x[9], C[2], y[5]);

    mux mux\_20(x[4], x[8], C[2], y[4]);

    mux mux\_19(x[3], x[7], C[2], y[3]);

    mux mux\_18(x[2], x[6], C[2], y[2]);

    mux mux\_17(x[1], x[5], C[2], y[1]);

    mux mux\_16(x[0], x[4], C[2], y[0]);

    //2bit shift right

    mux mux\_47(y[15], A[1], C[1], z[15]);

    mux mux\_46(y[14], A[0], C[1], z[14]);

    mux mux\_45(y[13], y[15], C[1], z[13]);

    mux mux\_44(y[12], y[14], C[1], z[12]);

    mux mux\_43(y[11], y[13], C[1], z[11]);

    mux mux\_42(y[10], y[12], C[1], z[10]);

    mux mux\_41(y[9], y[11], C[1], z[9]);

    mux mux\_40(y[8], y[10], C[1], z[8]);

    mux mux\_39(y[7], y[9], C[1], z[7]);

    mux mux\_38(y[6], y[8], C[1], z[6]);

    mux mux\_37(y[5], y[7], C[1], z[5]);

    mux mux\_36(y[4], y[6], C[1], z[4]);

    mux mux\_35(y[3], y[5], C[1], z[3]);

    mux mux\_34(y[2], y[4], C[1], z[2]);

    mux mux\_33(y[1], y[3], C[1], z[1]);

    mux mux\_32(y[0], y[2], C[1], z[0]);

    //1bit shift right

    mux mux\_63(z[15], A[0], C[0], O[15]);

    mux mux\_62(z[14], z[15], C[0], O[14]);

    mux mux\_61(z[13], z[14], C[0], O[13]);

    mux mux\_60(z[12], z[13], C[0], O[12]);

    mux mux\_59(z[11], z[12], C[0], O[11]);

    mux mux\_58(z[10], z[11], C[0], O[10]);

    mux mux\_57(z[9], z[10], C[0], O[9]);

    mux mux\_56(z[8], z[9], C[0], O[8]);

    mux mux\_55(z[7], z[8], C[0], O[7]);

    mux mux\_54(z[6], z[7], C[0], O[6]);

    mux mux\_53(z[5], z[6], C[0], O[5]);

    mux mux\_52(z[4], z[5], C[0], O[4]);

    mux mux\_51(z[3], z[4], C[0], O[3]);

    mux mux\_50(z[2], z[3], C[0], O[2]);

    mux mux\_49(z[1], z[2], C[0], O[1]);

    mux mux\_48(z[0], z[1], C[0], O[0]);

endmodule

**mux.v**

module mux(A0, A1, C, O);

    input A0, A1, C;

    output O;

   // assign O = (C == 1'b0) ? A0 : A1;

    assign O = ~C & A0 | C & A1;

endmodule

**testbench.v**

module tb();

    reg[15:0] A;

    reg[3:0] C;

    wire[15:0] O;

    barrel dut(.A(A), .C(C), .O(O));

    initial begin

      $dumpfile("dump.vcd");

      $dumpvars(0);

    end

    initial begin

        A <= 16'b1010101010101010;

        C <= 4'b0000;

        #100;

        A <= 16'b1010101010101010;

        C <= 4'b0001;

        #100;

        A <= 16'b1000000000000001;

        C <= 4'b0100;

        #100;

        A <= 16'b1000000100010001;

        C <= 4'b1000;

        #100;

        $finish;

    end

endmodule

**Waveforms:**

