

```
`timescale 1ns / 1ps
///////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/04/2022 04:45:07 PM
// Design Name:
// Module Name: count6U
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
///////////////////////////////

module count6U(
    input clk,
    input Up,
    input Dw,
    input LW,
    input [5:0] d,

    output [5:0] q
//    output UTC,
//    output DTC

);
//upper is active when all are 1
//    assign UTC = (q[5] & q[6] & q[3] & q[2] & q[1] & q[0]);
//lower active when all are 0
//    assign DTC = (~q[5]& ~q[4] & ~q[3] & ~q[2] & ~q[1] & ~q[0]);

    wire [5:0] out;
    wire [5:0] Upcnt;
    wire [4:0] Downcnt;
    wire cloud;

// making all the logic for the up and down count for the cloud
```

```

// broke down the logic into pieces to be easier to define
assign cloud = (Up ^ Dw) | LW;
// up count logic made to define the flip flops easier later which is FDRE
assign Upcnt[0] = (q[0] ^ cloud);
assign Upcnt[1] = q[1] ^ (cloud & q[0]);
assign Upcnt[2] = q[2] ^ (cloud & q[1] & q[0]);
assign Upcnt[3] = q[3] ^ (cloud & q[1] & q[2] & q[0]);
assign Upcnt[4] = q[4] ^ (cloud & q[1] & q[2] & q[0] & q[3]);
assign Upcnt[5] = q[5] ^ (cloud & q[1] & q[2] & q[0] & q[3] & q[4]);
// down count logic which is upcount but inversed
assign Downcnt[0] = q[1] ^ (cloud & ~q[0]);
assign Downcnt[1] = q[2] ^ (cloud & ~q[1] & ~q[0]);
assign Downcnt[2] = q[3] ^ (cloud & ~q[2] & ~q[1] & ~q[0]);
assign Downcnt[3] = q[4] ^ (cloud & ~q[3] & ~q[2] & ~q[1] & ~q[0]);
assign Downcnt[4] = q[5] ^ (cloud & ~q[4] & ~q[3] & ~q[2] & ~q[1] & ~q[0]);

//logic for the outputs of all the flip flops (4 bit flipflop)
assign out[5] = (((Upcnt[5] & Up & ~Dw) | (Downcnt[4] & ~Up & Dw)) & ~LW) | (LW
& d[5]));
assign out[4] = (((Upcnt[4] & Up & ~Dw) | (Downcnt[3] & ~Up & Dw)) & ~LW) | (LW
& d[4]));
assign out[3] = (((Upcnt[3] & Up & ~Dw) | (Downcnt[2] & ~Up & Dw)) & ~LW) | (LW
& d[3]));
assign out[2] = (((Upcnt[2] & Up & ~Dw) | (Downcnt[1] & ~Up & Dw)) & ~LW) | (LW
& d[2]));
assign out[1] = (((Upcnt[1] & Up & ~Dw) | (Downcnt[0] & ~Up & Dw)) & ~LW) | (LW
& d[1]));
assign out[0] = (((Upcnt[0] & Up & ~Dw) | (Upcnt[0] & ~Up & Dw)) & ~LW) | (LW &
d[0]));


```

```

FDRE #(.INIT(1'b0) ) flip5 (.C(clk), .CE(cloud), .D(out[5]), .Q(q[5]));
FDRE #(.INIT(1'b0) ) flip4 (.C(clk), .CE(cloud), .D(out[4]), .Q(q[4]));
FDRE #(.INIT(1'b0) ) flip3 (.C(clk), .CE(cloud), .D(out[3]), .Q(q[3]));
FDRE #(.INIT(1'b0) ) flip2 (.C(clk), .CE(cloud), .D(out[2]), .Q(q[2]));
FDRE #(.INIT(1'b0) ) flip1 (.C(clk), .CE(cloud), .D(out[1]), .Q(q[1]));
FDRE #(.INIT(1'b0) ) flip0 (.C(clk), .CE(cloud), .D(out[0]), .Q(q[0]));

```

```
endmodule
```