3bit_Sync_UpDown_Counter

```
/* During wave simulation in ModelSim,
Set the variable Clk as clock.
Force the value of Clr to 0 only for the first run.
After the first run, force the value to 1.
*/
module JK_FlipFlop(Q, Qb, J, K, Clk, Clr);
	input J, K, Clk, Clr;
	output Q, Qb;
	wire w1, w2, w3, w4, w5, Q1, Qb1, Clkb;

	nand(w1, J, Qb, Clk, Clr);
	nand(w2, K, Q, Clk);
	nand(Q1, w1, Qb1);
	nand(Qb1, w2, Q1, Clr);

	not(Clkb, Clk);

	nand(w3, Q1, Clkb);
	nand(w4, Qb1, Clkb);
	nand(Q, w3, Qb);
	nand(Qb, w4, Q, Clr);
endmodule

/* Dependencies: JK Flip-Flop */
/* During wave simulation in ModelSim,
Set the variable Clk as clock.
Force the value of Clr to 0 only for the first run.
```

```
After the first run, force the value to 1.
*/
module T_FlipFlop(Q, Qb, T, Clk, Clr);
        input T, Clk, Clr;
        output Q, Qb;

        JK_FlipFlop JKFF(Q, Qb, T, T, Clk, Clr);
Endmodule


/* Dependencies: JK Flip-Flop, T Flip-Flop */
/* During wave simulation in ModelSim,
Set the variable Clk as clock.
Force the value of Clr to 0 only for the first run.
After the first run, force the value to 1.
*/
module UpDown_Counter(Q, M, Clk, Clr);
        input M, Clk, Clr;
        output [2:0] Q;
        wire [2:0] Qb;
        wire Mb, and1, and2, and3, and4, t2, t3;

        not(Mb,M);
        T_FlipFlop TFF1(Q[0], Qb[0], 1, Clk, Clr);
        and(and1, Q[0], Mb);
        and(and2, Qb[0], M);
        or(t2, and1, and2);

        T_FlipFlop TFF2(Q[1], Qb[1], t2, Clk, Clr);
```

```
        and(and3, Q[1], and1);
        and(and4, Qb[1], and2);
        or(t3, and3, and4);

        T_FlipFlop TFF3(Q[2], Qb[2], t3, Clk, Clr);
Endmodule
```

4bit_Adder_Subtractor

```
module Half_Adder(sum, cout, a, b);
        input a, b;
        output sum, cout;

        xor(sum, a, b);
        and(cout, a, b);
endmodule

/* Dependencies: Half_Adder */
module Full_Adder(sum, cout, a, b, cin);
        input a, b, cin;
        output sum, cout;
        wire s1, cout1, cout2;

        Half_Adder HA1(s1, cout1, a, b);
        Half_Adder HA2(sum, cout2, s1, cin);
        or(cout,cout1,cout2);
endmodule

/* Dependencies: Half_Adder, Full_Adder */
module Four_bit_Full_Adder(cout, s, a, b, cin);
        input [3:0]a,b;
        input cin;
        output cout;
```

```
        output [3:0]s;
        wire c0,c1,c2;

        Full_Adder f1(s[0],c0,a[0],b[0],cin);
        Full_Adder f2(s[1],c1,a[1],b[1],c0);
        Full_Adder f3(s[2],c2,a[2],b[2],c1);
        Full_Adder f4(s[3],cout,a[3],b[3],c2);
Endmodule


/* Dependencies: Half_Adder, Full_Adder, 4bit_Full_Adder */
module Four_bit_Adder_Subtractor(cout, s, a, b, cin);
        // if cin=0, works as a adder. else, works as a subtractor
        input [3:0] a, b;
        input cin;
        output cout;
        output [3:0] s;
        wire [3:0] cin_xor_b;

        xor(cin_xor_b[0], b[0], cin);
        xor(cin_xor_b[1], b[1], cin);
        xor(cin_xor_b[2], b[2], cin);
        xor(cin_xor_b[3], b[3], cin);

        Four_bit_Full_Adder FFA(cout, s, a, cin_xor_b, cin);
endmodule

module Four_bit_Adder_Subtractor_tb;
        reg [3:0] a,b;
```

```
	reg cin;
	wire cout;
	wire [3:0] s;

	Four_bit_Adder_Subtractor AddSub(cout, s, a, b, cin);

	initial begin
		cin = 0; a = 4'b1000; b = 4'b0010; #50;
		cin = 0; a = 4'b1000; b = 4'b1000; #50;
		cin = 0; a = 4'b0010; b = 4'b1000; #50;
		cin = 0; a = 4'b0001; b = 4'b0111; #50;
		cin = 0; a = 4'b1010; b = 4'b1011; #50;
		cin = 0; a = 4'b1110; b = 4'b1111; #50;
		cin = 0; a = 4'b1010; b = 4'b1101; #50;
		cin = 1; a = 4'b1000; b = 4'b0010; #50;
		cin = 1; a = 4'b1000; b = 4'b1000; #50;
		cin = 1; a = 4'b0010; b = 4'b1000; #50;
		cin = 1; a = 4'b0001; b = 4'b0111; #50;
		cin = 1; a = 4'b1010; b = 4'b1011; #50;
		cin = 1; a = 4'b1110; b = 4'b1111; #50;
		cin = 1; a = 4'b1010; b = 4'b1101; #50;
	end
endmodule
```

4bit_BCD_Adder

```
module Half_Adder(sum, cout, a, b);
        input a, b;
        output sum, cout;

        xor(sum, a, b);
        and(cout, a, b);
endmodule


/* Dependencies: Half_Adder */
module Full_Adder(sum, cout, a, b, cin);
        input a, b, cin;
        output sum, cout;
        wire s1, cout1, cout2;

        Half_Adder HA1(s1, cout1, a, b);
        Half_Adder HA2(sum, cout2, s1, cin);
        or(cout,cout1,cout2);
endmodule


/* Dependencies: Half_Adder, Full_Adder */
module Four_bit_Full_Adder(cout, s, a, b, cin);
        input [3:0]a,b;
        input cin;
        output cout;
        output [3:0]s;
```

```
        wire c0,c1,c2;

        Full_Adder f1(s[0],c0,a[0],b[0],cin);
        Full_Adder f2(s[1],c1,a[1],b[1],c0);
        Full_Adder f3(s[2],c2,a[2],b[2],c1);
        Full_Adder f4(s[3],cout,a[3],b[3],c2);
Endmodule


/* Dependencies: Half_Adder, Full_Adder, 4bit_Full_Adder */
module BCD_Adder(cout, s, a, b, cin);
        input [3:0] a;
        input [3:0] b;
        input cin;
        output [3:0] s;
        output cout;
        reg cin2 = 1'b0;
        wire [3:0] s1, b2;
        wire and1, and2, or1, carry1, carry2;

        Four_bit_Full_Adder FFA1(carry1, s1, a, b, cin);

        and (and1, s1[3], s1[2]);
        and (and2, s1[3], s1[1]);
        or (or1, and1, and2);
        or(cout, carry1, or1);

        buf(b2[0], 1'b0);
        buf(b2[3], 1'b0);
```

```
        buf(b2[1], cout);
        buf(b2[2], cout);

        Four_bit_Full_Adder FFA2(carry2, s, s1, b2, cin2);

endmodule

module BCD_Adder_tb;
        reg [3:0] a, b;
        reg cin;
        wire cout;
        wire [3:0] s;

        BCD_Adder BA(cout, s, a, b, cin);

        initial begin
                a = 4'b0000; b = 4'b0000; cin = 1'b0; #50;
                a = 4'b0000; b = 4'b0001; cin = 1'b0; #50;
                a = 4'b0001; b = 4'b0001; cin = 1'b0; #50;
                a = 4'b0010; b = 4'b0001; cin = 1'b0; #50;
                a = 4'b0010; b = 4'b0010; cin = 1'b0; #50;
                a = 4'b0010; b = 4'b0011; cin = 1'b0; #50;
                a = 4'b0101; b = 4'b0001; cin = 1'b0; #50;
                a = 4'b0101; b = 4'b0010; cin = 1'b0; #50;
                a = 4'b1000; b = 4'b0000; cin = 1'b0; #50;
                a = 4'b0110; b = 4'b0011; cin = 1'b0; #50;
                a = 4'b1000; b = 4'b0001; cin = 1'b1; #50;
                a = 4'b1000; b = 4'b0010; cin = 1'b1; #50;
                a = 4'b1000; b = 4'b0011; cin = 1'b1; #50;
```

```
        a = 4'b1011; b = 4'b0001; cin = 1'b1; #50;
        a = 4'b0100; b = 4'b1001; cin = 1'b1; #50;
        a = 4'b0110; b = 4'b1000; cin = 1'b1; #50;
        a = 4'b1001; b = 4'b0110; cin = 1'b1; #50;
        a = 4'b1010; b = 4'b0110; cin = 1'b1; #50;
        a = 4'b1101; b = 4'b0100; cin = 1'b1; #50;
        a = 4'b1110; b = 4'b0100; cin = 1'b1; #50;
    end
endmodule
```

4bit_Sequential_Counter

```
/* During wave simulation in ModelSim,
Set the variable Clk as clock.
Force the value of Clr to 0 only for the first run.
After the first run, force the value to 1.
*/
module JK_FlipFlop(Q, Qb, J, K, Clk, Clr);
	input J, K, Clk, Clr;
	output Q, Qb;
	wire w1, w2, w3, w4, w5, Q1, Qb1, Clkb;

	nand(w1, J, Qb, Clk, Clr);
	nand(w2, K, Q, Clk);
	nand(Q1, w1, Qb1);
	nand(Qb1, w2, Q1, Clr);

	not(Clkb, Clk);

	nand(w3, Q1, Clkb);
	nand(w4, Qb1, Clkb);
	nand(Q, w3, Qb);
	nand(Qb, w4, Q, Clr);
endmodule


/* Dependencies: JK Flip-Flop */
/* During wave simulation in ModelSim,
Set the variable Clk as clock.
```

```
Force the value of Clr to 0 only for the first run.
After the first run, force the value to 1.
*/
module T_FlipFlop(Q, Qb, T, Clk, Clr);
        input T, Clk, Clr;
        output Q, Qb;

        JK_FlipFlop JKFF(Q, Qb, T, T, Clk, Clr);
Endmodule

/* Sequence: 0000 -> 1000 -> 1100 -> 1110 -> 1111 -> 0111 -> 0011 -> 0001 -> 0000 */
/* Dependencies: JK Flip-Flop, T Flip-Flop */
/* During wave simulation in ModelSim,
Set the variable Clk as clock.
Force the value of Clr variable to 0 only for the first run.
After the first run, Force the value to 1.
*/
module Sequential_Counter(Q, Clk, Clr);
        input Clk, Clr;
        output [3:0] Q;
        wire [3:0] Qb, t;

        xor(t[0], Q[0], Q[1]);
        T_FlipFlop TFF0(Q[0], Qb[0], t[0], Clk, Clr);

        xor(t[1], Q[1], Q[2]);
        T_FlipFlop TFF1(Q[1], Qb[1], t[1], Clk, Clr);

        xor(t[2], Q[2], Q[3]);
```

```
        T_FlipFlop TFF2(Q[2], Qb[2], t[2], Clk, Clr);


        xnor(t[3], Q[0], Q[3]);
        T_FlipFlop TFF3(Q[3], Qb[3], t[3], Clk, Clr);

Endmodule
```

BCD_to_Excess-3

```
module BCD_to_Excess3(e, b);
	input [3:0] b;
	output [3:0] e;
	wire w1, w2;			// w1=b[0]|b[1], w2=w1&b[2];

	not(e[0], b[0]);		// e[0] bit (LSB)
	xnor(e[1], b[0], b[1]); // e[1] bit
	or(w1, b[0], b[1]);
	xor(e[2], w1, b[2]);	// e[2] bit
	and(w2, w1, b[2]);
	or(e[3], w2, b[3]);			// e[3] bit (MSB)
endmodule

module BCD_to_Excess3_tb;
	reg [3:0] BCD;
	wire [3:0] Excess3;

	BCD_to_Excess3 excess3(.e(Excess3), .b(BCD));

	initial begin
		BCD = 4'b0000; #50;
		BCD = 4'b0001; #50;
		BCD = 4'b0010; #50;
		BCD = 4'b0011; #50;
		BCD = 4'b0100; #50;
		BCD = 4'b0101; #50;
```

```
                BCD = 4'b0110; #50;
                BCD = 4'b0111; #50;
                BCD = 4'b1000; #50;
                BCD = 4'b1001; #50;
                // input 4'b1010 to 4'b1111 were ignored
        end
endmodule
```

Excess-3_to_BCD

```
module Excess3_to_BCD(b, e);
	input [3:0] e;
	output [3:0] b;
	wire w1,w2;			// w1=e[0]|e[1], w2=w1&e[2];

	not(b[0], e[0]);
	xor(b[1], e[1], e[0]);
	and(w1, e[0], e[1]);
	xnor(b[2], w1, e[2]);
	or(w2, w1, e[2]);
	and(b[3], w2, e[3]);
endmodule

module Excess3_to_BCD_tb;
	reg [3:0] Excess3;
	wire [3:0] BCD;

	Excess3_to_BCD e2b(.b(BCD), .e(Excess3));

	initial begin
		// input 4'b0000 to 4'b0010 were ignored
		Excess3 = 4'b0011; #50;
		Excess3 = 4'b0100; #50;
		Excess3 = 4'b0101; #50;
		Excess3 = 4'b0110; #50;
		Excess3 = 4'b0111; #50;
```

```
                Excess3 = 4'b1000; #50;
                Excess3 = 4'b1001; #50;
                Excess3 = 4'b1010; #50;
                Excess3 = 4'b1011; #50;
                Excess3 = 4'b1100; #50;
                // input 4'b1101 to 4'b1111 were ignored
        end
endmodule
```