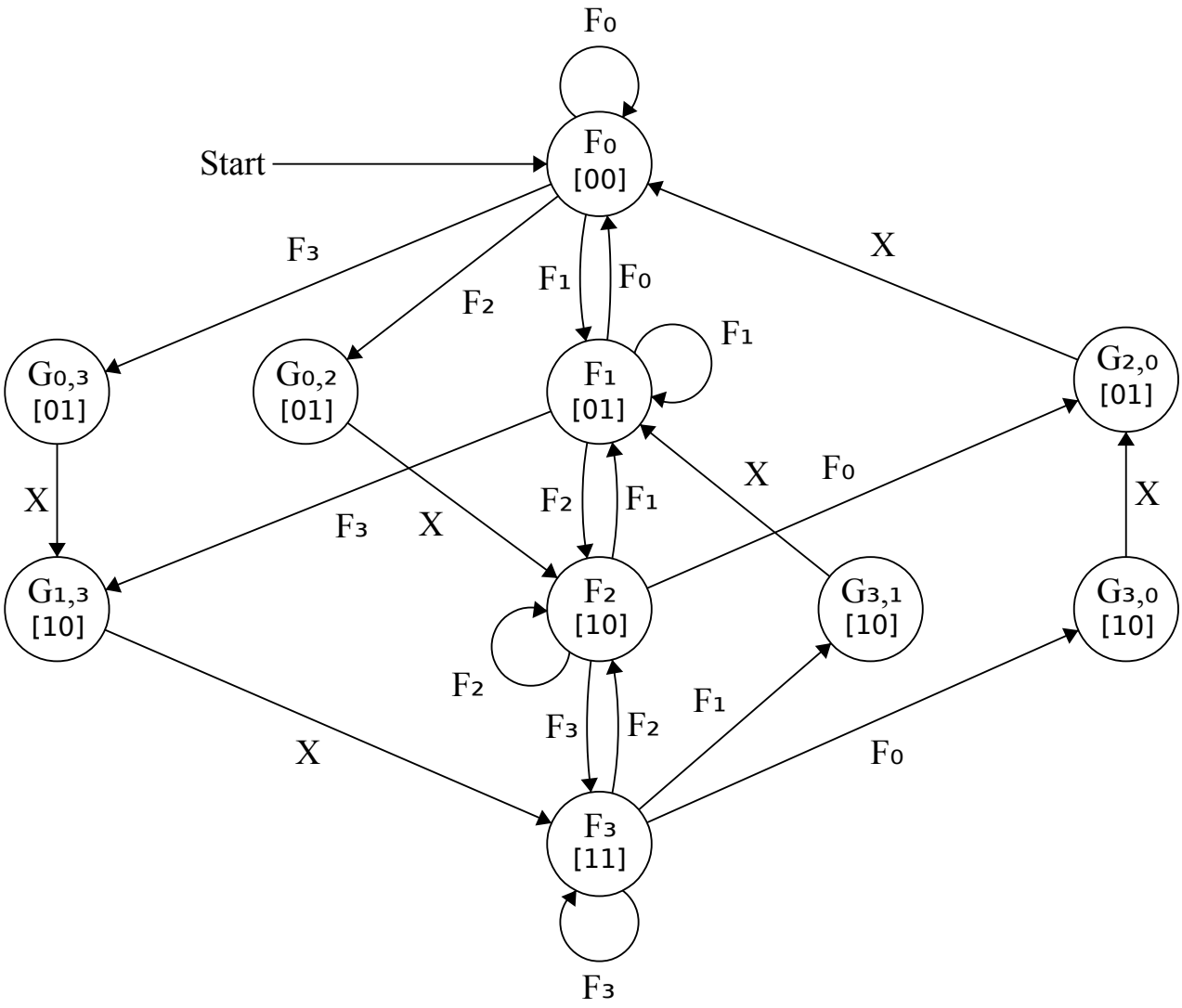
Moore FSM Diagram



States

F_0 = 0000

F_1 = 0001

F_2 = 0010

F_3 = 0011

$G_{0,2}$ = 0101

$G_{1,3}$ = 0110

$G_{2,0}$ = 1001

$G_{3,1}$ = 1010

$G_{0,3}$ = 1101

$G_{3,0}$ = 1110

State Transition Table

Current State				Input		Next State				Output	
Q_3	Q_2	Q_1	Q_0	B	A	Q_3^+	Q_2^+	Q_1^+	Q_0^+	Q_1	Q_0
0	0	0	0	0	0	0	0	0	0	0	0
				0	1	0	0	0	1	0	0
				1	*0*	***0***	***1***	***0***	***1***	0	0
				1	*1*	***1***	***1***	***0***	***1***	0	0
0	0	0	1	0	0	0	0	0	0	0	1
				0	1	0	0	0	1	0	1
				1	0	0	0	1	0	0	1
				1	*1*	***0***	***1***	***1***	***0***	0	1
0	0	1	0	*0*	*0*	***1***	***0***	***0***	***1***	1	0
				0	1	0	0	0	1	1	0
				1	0	0	0	1	0	1	0
				1	1	0	0	1	1	1	0
0	0	1	1	*0*	*0*	***1***	***1***	***1***	***0***	1	1
				0	*1*	***1***	***0***	***1***	***0***	1	1
				1	0	0	0	1	0	1	1
				1	1	0	0	1	1	1	1
0	***1***	***0***	***1***	X	X	0	0	1	0	0	1
0	***1***	***1***	***0***	X	X	0	0	1	1	1	0
1	***0***	***0***	***1***	X	X	*0*	*0*	*0*	*0*	0	1
1	***0***	***1***	***0***	X	X	0	0	0	1	1	0
1	***1***	***0***	***1***	X	X	***0***	***1***	***1***	***0***	0	1
1	***1***	***1***	***0***	X	X	***1***	***0***	***0***	***1***	1	0

Espresso Simplification

Input

```
#Elevator
.i 6
.o 4
```

Q	In	Q^+
3210	*BA*	*3210*
0000	00	0000
0000	01	0001
0000	10	0101
0000	11	1101
0001	00	0000
0001	01	0001
0001	10	0010
0001	11	0110
0010	00	1001
0010	01	0001
0010	10	0010
0010	11	0011
0011	00	1110
0011	01	1010
0011	10	0010
0011	11	0011
0101	--	0010
0110	--	0011
1001	--	0000
1010	--	0001
1101	--	0110
1110	--	1001

Output

```
#Elevator
.i 6
.o 4
.p 16
```

Q	In	D	
3210	*BA*	*3210*	
0011	00	0100	
000-	11	0100	
0000	11	1000	
0010	00	1001	*
1101	--	0100	
000-	01	0001	
1110	--	1000	
001-	11	0001	
0011	0-	1010	*
0000	1-	0101	*
00-1	1-	0010	
-101	--	0010	
0-10	1-	0010	
1-10	--	0001	
0110	--	0011	*
--10	-1	0001	

```
.e
```

Boolean Equations

$$D_3 = \overline{Q_3}\cdot\overline{Q_2}\cdot\overline{Q_1}\cdot\overline{Q_0}\cdot B\cdot A + \overline{Q_3}\cdot\overline{Q_2}\cdot Q_1\cdot\overline{Q_0}\cdot\overline{B}\cdot\overline{A} + Q_3\cdot Q_2\cdot Q_1\cdot\overline{Q_0} + \overline{Q_3}\cdot\overline{Q_2}\cdot Q_1\cdot Q_0\cdot\overline{B}$$

$$D_2 = \overline{Q_3}\cdot\overline{Q_2}\cdot Q_1\cdot Q_0\cdot\overline{B}\cdot\overline{A} + \overline{Q_3}\cdot\overline{Q_2}\cdot\overline{Q_1}\cdot B\cdot A + Q_3\cdot Q_2\cdot\overline{Q_1}\cdot Q_0 + \overline{Q_3}\cdot\overline{Q_2}\cdot\overline{Q_1}\cdot\overline{Q_0}\cdot B$$

$$D_1 = \overline{Q_3}\cdot\overline{Q_2}\cdot Q_1\cdot Q_0\cdot\overline{B} + \overline{Q_3}\cdot\overline{Q_2}\cdot Q_0\cdot B + Q_2\cdot\overline{Q_1}\cdot Q_0 + \overline{Q_3}\cdot Q_1\cdot\overline{Q_0}\cdot B + \overline{Q_3}\cdot Q_2\cdot Q_1\cdot\overline{Q_0}$$

$$D_0 = \overline{Q_3}\cdot\overline{Q_2}\cdot Q_1\cdot\overline{Q_0}\cdot\overline{B}\cdot\overline{A} + \overline{Q_3}\cdot\overline{Q_2}\cdot\overline{Q_1}\cdot\overline{B}\cdot A + \overline{Q_3}\cdot\overline{Q_2}\cdot Q_1\cdot B\cdot A + \overline{Q_3}\cdot\overline{Q_2}\cdot\overline{Q_1}\cdot\overline{Q_0}\cdot B + Q_3\cdot Q_1\cdot\overline{Q_0} + \overline{Q_3}\cdot Q_2\cdot Q_1\cdot\overline{Q_0} + Q_1\cdot\overline{Q_0}\cdot A$$

16 NOR Gates

3 – 3 input NOR gates
5 – 4 input NOR gates
5 – 5 input NOR gates
3 – 6 input NOR gates

4 OR Gates

2 – 4 input OR gates
1 – 5 input OR gate
1 – 7 input OR gate

32 Inverters

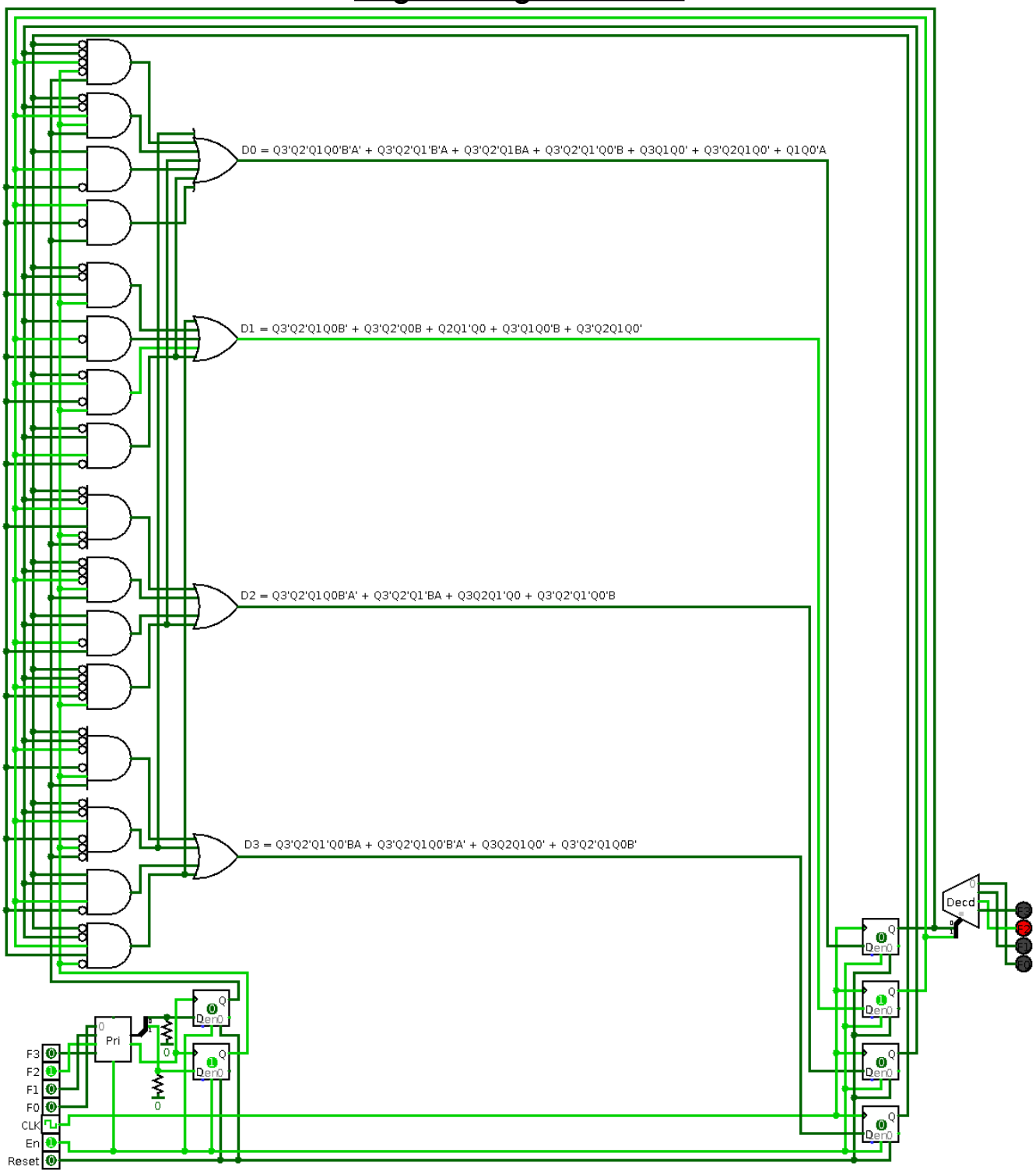
4 D-Flip-Flops

2 D-Latches

1 – 2-Select Bits Priority Encoder

1 – 2-Select Bits Decoder

Logisim Digital Circuit



Python MyHDL Code

```
from myhdl import always, always_comb, always_seq, Signal, ResetSignal, toVerilog,
toVHDL, delay, traceSignals, Simulation, now, intbv, concat

#Implementation

def elevator(clk, reset, en, F, D, Q, A, B, A_latch, B_latch, LED):

   @always_comb
   def encoder():
      if bool(F) and en:
         A.next = bool(F[3] or (F[1] and not F[2]))
         B.next = bool(F[2] or F[3])

   @always_comb
   def latch():
      if reset:
         A_latch.next = bool(False)
         B_latch.next = bool(False)
      elif bool(F) and en:
         A_latch.next = A
         B_latch.next = B

   @always_comb
   def logic():
      D.next = concat(   bool((not Q[3] and not Q[2] and not Q[1] and not Q[0] and
B_latch and A_latch) or (not Q[3] and not Q[2] and Q[1] and not Q[0] and not
B_latch and not A_latch) or (Q[3] and Q[2] and Q[1] and not Q[0]) or (not Q[3] and
not Q[2] and Q[1] and Q[0] and not B_latch)),
                         bool((not Q[3] and not Q[2] and Q[1] and Q[0] and not
B_latch and not A_latch) or (not Q[3] and not Q[2] and not Q[1] and B_latch and
A_latch) or (Q[3] and Q[2] and not Q[1] and Q[0]) or (not Q[3] and not Q[2] and not
Q[1] and not Q[0] and B_latch)),
                         bool((not Q[3] and not Q[2] and Q[1] and Q[0] and not
B_latch) or (not Q[3] and not Q[2] and Q[0] and B_latch) or (Q[2] and not Q[1] and
Q[0]) or (not Q[3] and Q[1] and not Q[0] and B_latch) or (not Q[3] and Q[2] and
Q[1] and not Q[0])),
                         bool((not Q[3] and not Q[2] and Q[1] and not Q[0] and not
B_latch and not A_latch) or (not Q[3] and not Q[2] and not Q[1] and not B_latch and
A_latch) or (not Q[3] and not Q[2] and Q[1] and B_latch and A_latch) or (not Q[3]
and not Q[2] and not Q[1] and not Q[0] and B_latch) or (Q[3] and Q[1] and not Q[0])
or (not Q[3] and Q[2] and Q[1] and not Q[0]) or (Q[1] and not Q[0] and
A_latch))   )

   @always_seq(clk.posedge, reset)
   def dff():
      if en:
         Q.next = D

   @always_comb
   def decoder():
      LED.next = concat(   bool(Q[1] and Q[0]),
                           bool(Q[1] and not Q[0]),
                           bool(not Q[1] and Q[0]),
                           bool(not Q[1] and not Q[0])   )

   return encoder, latch, logic, dff, decoder
```

Python MyHDL Verilog and VHDL Conversion Code

```
#Convert to Verilog

def convert():
   clk = Signal(bool(False))
   reset = ResetSignal(bool(False), bool(True), async=True)
   en = Signal(bool(True))
   F = Signal(intbv(0b0, 0b0, 0b10000))
   D = Signal(intbv(0b0, 0b0, 0b10000))
   Q = Signal(intbv(0b0, 0b0, 0b10000))
   A = Signal(bool(False))
   B = Signal(bool(False))
   A_latch = Signal(bool(False))
   B_latch = Signal(bool(False))
   LED = Signal(intbv(0b0, 0b0, 0b10000))
   toVerilog.timescale = "100ms/1ms"
   toVerilog(elevator, clk, reset, en, F, D, Q, A, B, A_latch, B_latch, LED)
   toVHDL(elevator, clk, reset, en, F, D, Q, A, B, A_latch, B_latch, LED)

convert()
```

Python MyHDL Simulation Code

```
#Simulate

def test_elevator():
   clk = Signal(bool(False))
   reset = ResetSignal(bool(False), bool(True), async=True)
   en = Signal(bool(True))
   F = Signal(intbv(0b0, 0b0, 0b10000))
   D = Signal(intbv(0b0, 0b0, 0b10000))
   Q = Signal(intbv(0b0, 0b0, 0b10000))
   A = Signal(bool(False))
   B = Signal(bool(False))
   A_latch = Signal(bool(False))
   B_latch = Signal(bool(False))
   LED = Signal(intbv(0b0, 0b0, 0b10000))
   elevator_inst = elevator(clk, reset, en, F, D, Q, A, B, A_latch, B_latch, LED)

   @always(delay(10))
   def clkgen():
      clk.next = not clk

   @always(delay(1))
   def stimulus():
      if now() == 45:
         F.next = Signal(intbv(0b0100))
      elif now() == 46:
         F.next = Signal(intbv(0b0000))
      elif now() == 60:
         F.next = Signal(intbv(0b0001))
      elif now() == 61:
         F.next = Signal(intbv(0b0000))
      elif now() == 150:
         F.next = Signal(intbv(0b0010))
      elif now() == 200:
         F.next = Signal(intbv(0b0011))
      elif now() == 250:
         F.next = Signal(intbv(0b0100))
      elif now() == 300:
         F.next = Signal(intbv(0b0101))
      elif now() == 350:
         F.next = Signal(intbv(0b0110))
      elif now() == 400:
         F.next = Signal(intbv(0b0111))
      elif now() == 450:
         F.next = Signal(intbv(0b1000))
      elif now() == 500:
         F.next = Signal(intbv(0b1001))
      elif now() == 550:
         F.next = Signal(intbv(0b1010))
      elif now() == 600:
         F.next = Signal(intbv(0b1011))
      elif now() == 650:
         F.next = Signal(intbv(0b1100))
         #reset.next = 0b1
      elif now() == 700:
         F.next = Signal(intbv(0b1101))
      elif now() == 750:
         F.next = Signal(intbv(0b1110))
      elif now() == 800:
         F.next = Signal(intbv(0b0001))
      elif now() == 850:
         F.next = Signal(intbv(0b1000))
         #reset.next = 0b0
      elif now() == 900:
         F.next = Signal(intbv(0b0010))
      elif now() == 950:
         F.next = Signal(intbv(0b1000))
         #en.next = 0b0
      elif now() == 1000:
         F.next = Signal(intbv(0b0001))
      elif now() == 1050:
         F.next = Signal(intbv(0b0100))
      elif now() == 1100:
         F.next = Signal(intbv(0b0001))
      elif now() == 1150:
         F.next = Signal(intbv(0b0100))
      elif now() == 1200:
         F.next = Signal(intbv(0b0000))
      elif now() == 1240:
         F.next = Signal(intbv(0b0000))
         #en.next = 0b1
      elif now() == 1360:
         F.next = Signal(intbv(0b0001))
      elif now() == 1370:
         F.next = Signal(intbv(0b1001))

   return elevator_inst, clkgen, stimulus#, reset_test, en_test


def simulate(timesteps):
   traceSignals.timescale = "100ms"
   tb = traceSignals(test_elevator)
   sim = Simulation(tb)
   sim.run(timesteps)

simulate(1500)
```

Verilog Code

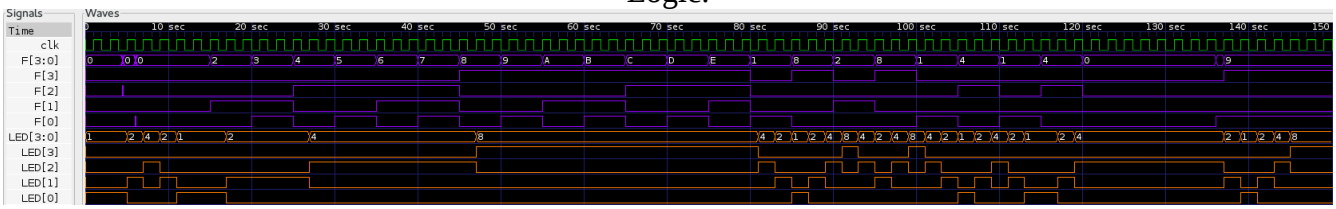
```
// File: elevator.v
// Generated by MyHDL 0.8.1
// Date: Sun Jun 14 22:11:56 2015

`timescale 100ms/1ms

module elevator (
    clk,
    reset,
    en,
    F,
    D,
    Q,
    A,
    B,
    A_latch,
    B_latch,
    LED
);

input clk;
input reset;
input en;
input [3:0] F;
output [3:0] D;
wire [3:0] D;
output [3:0] Q;
reg [3:0] Q;
output A;
reg A;
output B;
reg B;
output A_latch;
reg A_latch;
output B_latch;
reg B_latch;
output [3:0] LED;
wire [3:0] LED;

always @(en, F) begin: ELEVATOR_ENCODER
    if (((F != 0) && en)) begin
        A = ((F[3] || (F[1] && (!F[2]))) != 0);
        B = ((F[2] || F[3]) != 0);
    end
end

always @(reset, A, B, en, F) begin: ELEVATOR_LATCH
    if (reset) begin
        A_latch = (1'b0 != 0);
        B_latch = (1'b0 != 0);
    end
    else if (((F != 0) && en)) begin
        A_latch = A;
        B_latch = B;
    end
end

assign D = {(((((!Q[3]) && (!Q[2]) && (!Q[1]) && (!Q[0]) && B_latch && A_latch) || ((!Q[3]) && (!Q[2]) && Q[1] && (!
Q[0]) && (!B_latch) && (!A_latch)) || (Q[3] && Q[2] && Q[1] && (!Q[0])) || ((!Q[3]) && (!Q[2]) && Q[1] && Q[0] && (!
B_latch))) != 0), ((((!Q[3]) && (!Q[2]) && Q[1] && Q[0] && (!B_latch) && (!A_latch)) || ((!Q[3]) && (!Q[2]) && (!Q[1])
&& B_latch && A_latch) || (Q[3] && Q[2] && (!Q[1]) && Q[0]) || ((!Q[3]) && (!Q[2]) && (!Q[1]) && (!Q[0]) && B_latch))
!= 0), ((((!Q[3]) && (!Q[2]) && Q[1] && Q[0] && (!B_latch)) || ((!Q[3]) && (!Q[2]) && Q[0] && B_latch) || (Q[2] && (!
Q[1]) && Q[0]) || ((!Q[3]) && Q[1] && (!Q[0]) && B_latch) || ((!Q[3]) && Q[2] && Q[1] && (!Q[0]))) != 0), ((((!Q[3])
&& (!Q[2]) && Q[1] && (!Q[0]) && (!B_latch) && (!A_latch)) || ((!Q[3]) && (!Q[2]) && (!Q[1]) && (!B_latch) && A_latch)
|| ((!Q[3]) && (!Q[2]) && Q[1] && B_latch && A_latch) || ((!Q[3]) && (!Q[2]) && (!Q[1]) && (!Q[0]) && B_latch) ||
(Q[3] && Q[1] && (!Q[0])) || ((!Q[3]) && Q[2] && Q[1] && (!Q[0])) || (Q[1] && (!Q[0]) && A_latch)) != 0)};

always @(posedge clk, posedge reset) begin: ELEVATOR_DFF
    if (reset == 1) begin
        Q <= 0;
    end
    else begin
        if (en) begin
            Q <= D;
        end
    end
end

assign LED = {((Q[1] && Q[0]) != 0), ((Q[1] && (!Q[0])) != 0), ((((!Q[1]) && Q[0]) != 0), (((!Q[1]) && (!Q[0])) != 0)};

endmodule
```

VHDL Code

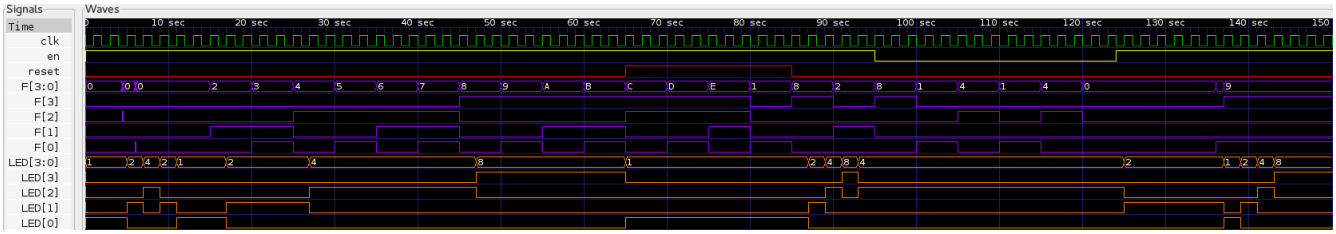
```
-- File: elevator.vhd
-- Generated by MyHDL 0.8.1
-- Date: Sun Jun 14 22:11:56 2015

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use std.textio.all;

use work.pck_myhdl_081.all;

entity elevator is
    port (
        clk: in std_logic;
        reset: in std_logic;
        en: in std_logic;
        F: in unsigned(3 downto 0);
        D: inout unsigned(3 downto 0);
        Q: inout unsigned(3 downto 0);
        A: inout std_logic;
        B: inout std_logic;
        A_latch: inout std_logic;
        B_latch: inout std_logic;
        LED: out unsigned(3 downto 0)
    );
end entity elevator;

architecture MyHDL of elevator is

begin

ELEVATOR_ENCODER: process (en, F) is
begin
    if (bool(F) and bool(en)) then
        A <= stdl(bool(F(3)) or (bool(F(1)) and (not bool(F(2)))));
        B <= stdl(bool(F(2)) or bool(F(3)));
    end if;
end process ELEVATOR_ENCODER;

ELEVATOR_LATCH: process (reset, A, B, en, F) is
begin
    if bool(reset) then
        A_latch <= '0';
        B_latch <= '0';
    elsif (bool(F) and bool(en)) then
        A_latch <= A;
        B_latch <= B;
    end if;
end process ELEVATOR_LATCH;

D <= unsigned'((((not bool(Q(3))) and (not bool(Q(2))) and (not bool(Q(1))) and (not bool(Q(0))) and bool(B_latch) and
bool(A_latch)) or ((not bool(Q(3))) and (not bool(Q(2))) and bool(Q(1)) and (not bool(Q(0))) and (not bool(B_latch)) and
(not bool(A_latch))) or (bool(Q(3)) and bool(Q(2)) and bool(Q(1)) and (not bool(Q(0)))) or ((not bool(Q(3))) and (not
bool(Q(2))) and bool(Q(1)) and bool(Q(0)) and (not bool(B_latch)))) & (((not bool(Q(3))) and (not bool(Q(2))) and
bool(Q(1)) and bool(Q(0)) and (not bool(B_latch)) and (not bool(A_latch))) or ((not bool(Q(3))) and (not bool(Q(2))) and
(not bool(Q(1))) and bool(B_latch) and bool(A_latch)) or (bool(Q(3)) and bool(Q(2)) and (not bool(Q(1))) and bool(Q(0)))
or ((not bool(Q(3))) and (not bool(Q(2))) and (not bool(Q(1))) and (not bool(Q(0))) and bool(B_latch))) & (((not
bool(Q(3))) and (not bool(Q(2))) and bool(Q(1)) and bool(Q(0)) and (not bool(B_latch))) or ((not bool(Q(3))) and (not
bool(Q(2))) and bool(Q(0)) and bool(B_latch)) or (bool(Q(2)) and (not bool(Q(1))) and bool(Q(0))) or ((not bool(Q(3)))
and bool(Q(1)) and (not bool(Q(0))) and bool(B_latch)) or ((not bool(Q(3))) and bool(Q(2)) and bool(Q(1)) and (not
bool(Q(0))))) & (((not bool(Q(3))) and (not bool(Q(2))) and bool(Q(1)) and (not bool(Q(0))) and (not bool(B_latch)) and
(not bool(A_latch))) or ((not bool(Q(3))) and (not bool(Q(2))) and (not bool(Q(1))) and (not bool(B_latch)) and
bool(A_latch)) or ((not bool(Q(3))) and (not bool(Q(2))) and bool(Q(1)) and bool(B_latch) and bool(A_latch)) or ((not
bool(Q(3))) and (not bool(Q(2))) and (not bool(Q(1))) and (not bool(Q(0))) and bool(B_latch)) or (bool(Q(3)) and
bool(Q(1)) and (not bool(Q(0)))) or ((not bool(Q(3))) and bool(Q(2)) and bool(Q(1)) and (not bool(Q(0)))) or (bool(Q(1))
and (not bool(Q(0))) and bool(A_latch))));

ELEVATOR_DFF: process (clk, reset) is
begin
    if (reset = '1') then
        Q <= to_unsigned(0, 4);
    elsif rising_edge(clk) then
        if bool(en) then
            Q <= D;
        end if;
    end if;
end process ELEVATOR_DFF;

LED <= unsigned'((bool(Q(1)) and bool(Q(0))) & (bool(Q(1)) and (not bool(Q(0)))) & ((not bool(Q(1))) and bool(Q(0))) &
((not bool(Q(1))) and (not bool(Q(0)))));

end architecture MyHDL;
```

Waveform Viewer

Logic:



Reset and en:



The design can be improved by having the output going back and forth between extremes when multiple buttons are selected simultaneously, optimally by first going to the closest floor. This can be implemented with an additional FSM in place of the priority encoder.