Or Gate Logic

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity OrGate is
	Port (
		A : in STD_LOGIC;
		B : in STD_LOGIC;
		Y : out STD_LOGIC
	 );
end OrGate;

architecture Behavioral of OrGate is
begin
	Y <= A or B;
end Behavioral;
```

And Gate Logic

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity AndGate is
	Port (
		A : in STD_LOGIC;
		B : in STD_LOGIC;
		Y : out STD_LOGIC
	);
end AndGate;

architecture Behavioral of AndGate is
begin
```

```
        Y <= A and B;
end Behavioral;
```

Not Gate Logic

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity NotGate is
        Port (
                A : in STD_LOGIC;
                Y : out STD_LOGIC
        );
end NotGate;
architecture Behavioral of NotGate is
begin
        Y <= not A;
end Behavioral;
```

Clock

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ClockGenerator is
   Port (
      clk_out : out STD_LOGIC;
      reset   : in STD_LOGIC;
      enable  : in STD_LOGIC;
      clk_period : in INTEGER;
```

```
  );
end ClockGenerator;

architecture Behavioral of ClockGenerator is
  signal clk_int : STD_LOGIC := '0';
  signal count   : INTEGER := 0;

begin
  process(reset, enable)
  begin
    if reset = '1' then
      clk_int <= '0';
      count <= 0;
    elsif enable = '1' then
      count <= count + 1;
      if count = clk_period / 2 then
        clk_int <= not clk_int;
        count <= 0;
      end if;
    end if;
  end process;

  clk_out <= clk_int;

end Behavioral;
```

JKFlip-Flop

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity JKFlipFlop is
    Port (
        J    : in  STD_LOGIC;
        K    : in  STD_LOGIC;
        clk  : in  STD_LOGIC;
        reset : in  STD_LOGIC;
        Q    : out STD_LOGIC;
        Qn   : out STD_LOGIC
    );
end JKFlipFlop;

architecture Behavioral of JKFlipFlop is
begin
    process(clk, reset)
    begin
        if reset = '1' then
            Q <= '0';
            Qn <= '1';
        elsif rising_edge(clk) then
            if J = '1' and K = '0' then
                Q <= '1';
            elsif J = '0' and K = '1' then
                Q <= '0';
            elsif J = '1' and K = '1' then
                Q <= not Q;
            end if;
        end if;
        Qn <= not Q;  -- Inverted output
```

```
    end process;

end Behavioral;
```

Main File

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MainCounter is
    Port (
        input1  : in  STD_LOGIC;
        input2  : in  STD_LOGIC;
        clk     : in  STD_LOGIC;
        reset   : in  STD_LOGIC;
        output1 : out STD_LOGIC;
        output2 : out STD_LOGIC;
        output3 : out STD_LOGIC
    );
end MainCounter;

architecture Behavioral of MainCounter is
    signal J1, K1 : STD_LOGIC;
    signal Q1     : STD_LOGIC;
    signal Q1n    : STD_LOGIC;
    signal J2, K2 : STD_LOGIC;
    signal Q2     : STD_LOGIC;
    signal Not_input2 : STD_LOGIC;

    component JKFlipFlop
```

```
        Port (
            J    : in  STD_LOGIC;
            K    : in  STD_LOGIC;
            clk  : in  STD_LOGIC;
            reset : in  STD_LOGIC;
            Q    : out STD_LOGIC;
            Qn   : out STD_LOGIC
        );
    end component;

begin
    output1 <= input2;
    Not_input2 <= not input2;

    J1 <= input1 and Not_input2;
    K1 <= input1 and Not_input2;

    FF1: JKFlipFlop port map (J1, K1, clk, reset, Q1, Q1n);
    output2 <= Q1;

    output3 <= Not_input2 and Q1;

    J2 <= Q1;
    K2 <= Q1;

    FF2: JKFlipFlop port map (J2, K2, clk, reset, Q2, open);
    output2 <= Q2;

end Behavioral;
```

Note: This code can be used with any digital design development board with the appropriate display drivers added instead of output and the appropriate xdc file