Home
The Go Board
FPGA 101
VHDL
Verilog
YouTube Channel
GitHub
Patreon *NEW*

Tutorial: Your First FPGA Program: An LED Blinker

Part 1: Design of VHDL or Verilog

This tutorial shows the construction of VHDL and Verilog code that blinks an LED at a specified frequency. Both VHDL and Verilog are shown, and you can choose which you want to learn first. Whenever design code is written the FPGA designer needs to ensure that it works the way that it was intended. Despite your best efforts, there will always be mistakes in your initial design. The best way to find these mistakes is in a simulation environment. This tutorial is broken up into 2 stages:

1. Design of HDL
2. Simulation of HDL

Both of these steps are crucial for successful FPGA development. Sometimes FPGA designers who are pressed for time will try to skip step two, the simulation of their code. However this is an extremely important step! Without proper simulation you will be forced to debug your code on hardware which can be a very difficult and time consuming endeavour.

Project Requirements:

Design HDL code that will blink an LED at a specified frequency of 100 Hz, 50 Hz, 10 Hz, or 1 Hz. For each of the blink frequencies, the LED will be set to 50% duty cycle (it will be on half the time). The LED frequency will be chosen via two switches which are inputs to the FPGA. There is an additional switch called LED_EN that needs to be '1' to turn on the LED. The FPGA will be driven by a 25 MHz oscillator.

Let's first draw the truth table for the frequency selector:

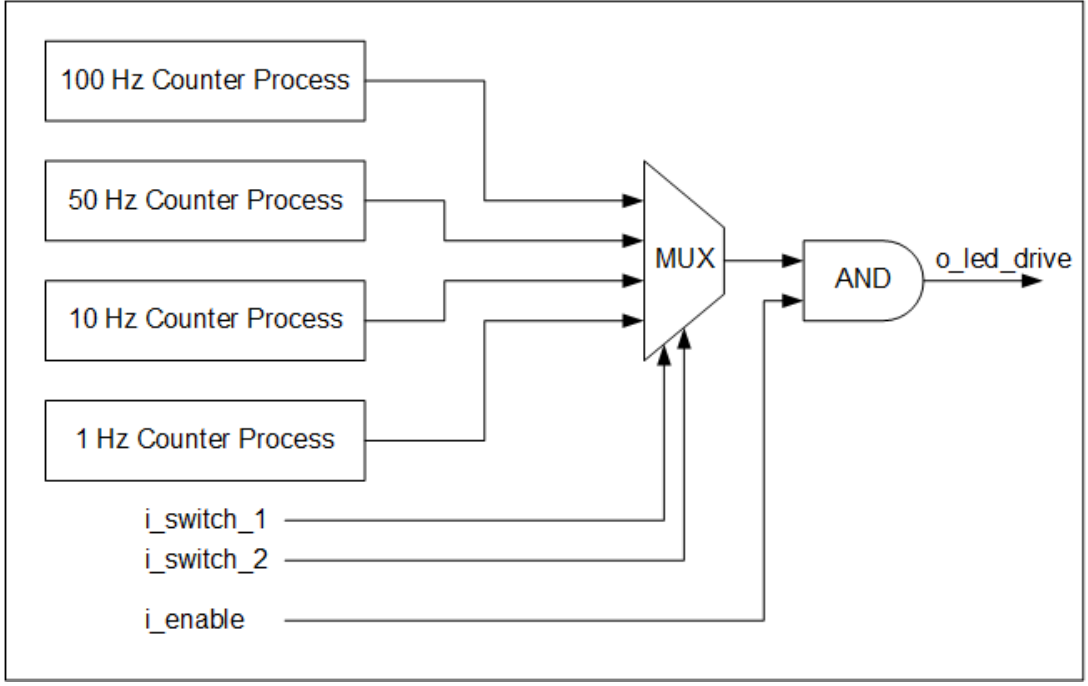
Enable	Switch 1	Switch 2	LED Drive Frequency
0	-	-	(disabled)
1	0	0	100 Hz
1	0	1	50 Hz
1	1	0	10 Hz
1	1	1	1 Hz

For this to work correctly there will be 4 inputs and 1 output. The signals will be:

Signal Name	Direction	Description
i_clock	Input	25 MHz Clock
i_enable	Input	The Enable Switch (Logic 0 = No LED Drive)
i_switch_1	Input	Switch 1 in the Truth Table above
i_switch_2	Input	Switch 2 in the Truth Table above
o_led_drive	Output	The signal that drives the LED

For the design there are four counter processes that run concurrently. This means that they are all running at the exact same time. Their job is to keep track of the number of clock pulses seen for each of the different frequencies. Even if the switches are not selecting that particular frequency, the counters are still running! This is the beauty of Hardware Design and concurrency. Everything runs all the time! It can be challenging to understand this initially, but it is the core concept that you need to master.

The switches only serve to select which output to use. They create what is known as a multiplexer. A multiplexer or mux for short is a selector that will select one of a number of inputs to propagate or pass to the output. It is a combinatorial piece of logic, meaning that it does not require a clock to operate. Below is a block diagram of the design. Spend some time thinking about how you might implement this design. Try writing the code yourself. The way that I chose to do can be found below.



Block Diagram - LED Blink Program

VHDL code for the design, tutorial_led_blink.vhd:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity tutorial_led_blink is
6    port (
7      i_clock      : in  std_logic;
8      i_enable     : in  std_logic;
9      i_switch_1   : in  std_logic;
10     i_switch_2   : in  std_logic;
11     o_led_drive  : out std_logic
12     );
13 end tutorial_led_blink;
14
15 architecture rtl of tutorial_led_blink is
16
17   -- Constants to create the frequencies needed:
18   -- Formula is: (25 MHz / 100 Hz * 50% duty cycle)
19   -- So for 100 Hz: 25,000,000 / 100 * 0.5 = 125,000
20   constant c_CNT_100HZ : natural := 125000;
21   constant c_CNT_50HZ  : natural := 250000;
22   constant c_CNT_10HZ  : natural := 1250000;
23   constant c_CNT_1HZ   : natural := 12500000;
24
25
26   -- These signals will be the counters:
27   signal r_CNT_100HZ : natural range 0 to c_CNT_100HZ;
28   signal r_CNT_50HZ  : natural range 0 to c_CNT_50HZ;
29   signal r_CNT_10HZ  : natural range 0 to c_CNT_10HZ;
30   signal r_CNT_1HZ   : natural range 0 to c_CNT_1HZ;
31
32   -- These signals will toggle at the frequencies needed:
33   signal r_TOGGLE_100HZ : std_logic := '0';
34   signal r_TOGGLE_50HZ  : std_logic := '0';
35   signal r_TOGGLE_10HZ  : std_logic := '0';
36   signal r_TOGGLE_1HZ   : std_logic := '0';
37
38   -- One bit select wire.
39   signal w_LED_SELECT : std_logic;
40
41 begin
42
43   -- All processes toggle a specific signal at a different frequency.
44   -- They all run continuously even if the switches are
45   -- not selecting their particular output.
46
47   p_100_HZ : process (i_clock) is
```

```
  begin
    if rising_edge(i_clock) then
      if r_CNT_100HZ = c_CNT_100HZ-1 then  -- -1, since counter starts at 0
        r_TOGGLE_100HZ <= not r_TOGGLE_100HZ;
        r_CNT_100HZ    <= 0;
      else
        r_CNT_100HZ <= r_CNT_100HZ + 1;
      end if;
    end if;
  end process p_100_HZ;


  p_50_HZ : process (i_clock) is
  begin
    if rising_edge(i_clock) then
      if r_CNT_50HZ = c_CNT_50HZ-1 then  -- -1, since counter starts at 0
        r_TOGGLE_50HZ <= not r_TOGGLE_50HZ;
        r_CNT_50HZ    <= 0;
      else
        r_CNT_50HZ <= r_CNT_50HZ + 1;
      end if;
    end if;
  end process p_50_HZ;


  p_10_HZ : process (i_clock) is
  begin
    if rising_edge(i_clock) then
      if r_CNT_10HZ = c_CNT_10HZ-1 then  -- -1, since counter starts at 0
        r_TOGGLE_10HZ <= not r_TOGGLE_10HZ;
        r_CNT_10HZ    <= 0;
      else
        r_CNT_10HZ <= r_CNT_10HZ + 1;
      end if;
    end if;
  end process p_10_HZ;


  p_1_HZ : process (i_clock) is
  begin
    if rising_edge(i_clock) then
      if r_CNT_1HZ = c_CNT_1HZ-1 then  -- -1, since counter starts at 0
        r_TOGGLE_1HZ <= not r_TOGGLE_1HZ;
        r_CNT_1HZ    <= 0;
      else
        r_CNT_1HZ <= r_CNT_1HZ + 1;
      end if;
    end if;
  end process p_1_HZ;


  -- Create a multiplexor based on switch inputs
  w_LED_SELECT <= r_TOGGLE_100HZ when (i_switch_1 = '0' and i_switch_2 = '0') else
                  r_TOGGLE_50HZ  when (i_switch_1 = '0' and i_switch_2 = '1') else
                  r_TOGGLE_10HZ  when (i_switch_1 = '1' and i_switch_2 = '0') else
                  r_TOGGLE_1HZ;


  -- Only allow o_led_drive to drive when i_enable is high (and gate).
  o_led_drive <= w_LED_SELECT and i_enable;

end rtl;
```

Verilog code for the design, tutorial_led_blink.v:

```
module tutorial_led_blink
  (
   i_clock,
   i_enable,
   i_switch_1,
   i_switch_2,
   o_led_drive
   );

  input i_clock;
  input i_enable;
  input i_switch_1;
```

```
13    input i_switch_2;
14    output o_led_drive;
15
16    // Constants (parameters) to create the frequencies needed:
17    // Input clock is 25 kHz, chosen arbitrarily.
18    // Formula is: (25 kHz / 100 Hz * 50% duty cycle)
19    // So for 100 Hz: 25,000 / 100 * 0.5 = 125
20    parameter c_CNT_100HZ = 125;
21    parameter c_CNT_50HZ  = 250;
22    parameter c_CNT_10HZ  = 1250;
23    parameter c_CNT_1HZ   = 12500;
24
25    // These signals will be the counters:
26    reg [31:0] r_CNT_100HZ = 0;
27    reg [31:0] r_CNT_50HZ = 0;
28    reg [31:0] r_CNT_10HZ = 0;
29    reg [31:0] r_CNT_1HZ = 0;
30
31    // These signals will toggle at the frequencies needed:
32    reg        r_TOGGLE_100HZ = 1'b0;
33    reg        r_TOGGLE_50HZ  = 1'b0;
34    reg        r_TOGGLE_10HZ  = 1'b0;
35    reg        r_TOGGLE_1HZ   = 1'b0;
36
37    // One bit select
38    reg        r_LED_SELECT;
39    wire       w_LED_SELECT;
40
41
42  begin
43
44    // All always blocks toggle a specific signal at a different frequency.
45    // They all run continuously even if the switches are
46    // not selecting their particular output.
47
48    always @ (posedge i_clock)
49      begin
50        if (r_CNT_100HZ == c_CNT_100HZ-1) // -1, since counter starts at 0
51          begin
52            r_TOGGLE_100HZ <= !r_TOGGLE_100HZ;
53            r_CNT_100HZ    <= 0;
54          end
55        else
56          r_CNT_100HZ <= r_CNT_100HZ + 1;
57      end
58
59
60    always @ (posedge i_clock)
61      begin
62        if (r_CNT_50HZ == c_CNT_50HZ-1) // -1, since counter starts at 0
63          begin
64            r_TOGGLE_50HZ <= !r_TOGGLE_50HZ;
65            r_CNT_50HZ    <= 0;
66          end
67        else
68          r_CNT_50HZ <= r_CNT_50HZ + 1;
69      end
70
71
72    always @ (posedge i_clock)
73      begin
74        if (r_CNT_10HZ == c_CNT_10HZ-1) // -1, since counter starts at 0
75          begin
76            r_TOGGLE_10HZ <= !r_TOGGLE_10HZ;
77            r_CNT_10HZ    <= 0;
78          end
79        else
80          r_CNT_10HZ <= r_CNT_10HZ + 1;
81      end
82
83
84    always @ (posedge i_clock)
85      begin
86        if (r_CNT_1HZ == c_CNT_1HZ-1) // -1, since counter starts at 0
87          begin
88            r_TOGGLE_1HZ <= !r_TOGGLE_1HZ;
89            r_CNT_1HZ    <= 0;
90          end
91        else
```

```
92            r_CNT_1HZ <= r_CNT_1HZ + 1;
93        end
94
95      // Create a multiplexer based on switch inputs
96      always @ (*)
97      begin
98        case ({i_switch_1, i_switch_2}) // Concatenation Operator { }
99          2'b11 : r_LED_SELECT <= r_TOGGLE_1HZ;
100         2'b10 : r_LED_SELECT <= r_TOGGLE_10HZ;
101         2'b01 : r_LED_SELECT <= r_TOGGLE_50HZ;
102         2'b00 : r_LED_SELECT <= r_TOGGLE_100HZ;
103       endcase
104     end
105
106     assign o_led_drive = r_LED_SELECT & i_enable;
107
108     // Alternative way to design multiplexer (same as above):
109     // More compact, but harder to read, especially to those new to Verilog
110     // assign w_LED_SELECT = i_switch_1 ? (i_switch_2 ? r_TOGGLE_1HZ : r_TOGGLE_10HZ)
111                                         (i_switch_2 ? r_TOGGLE_50HZ : r_TOGGLE_100H
112     // assign o_led_drive = w_LED_SELECT & i_enable;
113
114
115 end
116
117 endmodule
```

Next Step: Simulating this design in VHDL or Verilog!

Help Me Make Great Content! **Support me on Patreon!** **Buy a Go Board!**