EXPLORING XILINX VIVADO IDE AND ZEDBOARD

Ridge Tejuco

California State University, Northridge, Electrical and Computer Engineering Department
ridge.tejuco.881@my.csun.edu

Abstract:

The Zedboard is an fpga development board using the Xilinx Zynq-7000 SoC. The Zedboard provides a wide range of functions and interfaces that enable developers to rapidly prototype circuit designs. This experiment covers the design and implementation of a logical circuit onto a Zedboard. The design consists of 8 LEDs controlled by various types of 8 bit counters. The circuit was coded in VHDL and simulated in Vivado. Various test benches were constructed resulting in the expected sequences for each counter. Due to restrictions in place for Covid-19, the physical lab and Zedboard is unavailable and only the simulation and VHDL code is included in the report.

Keywords:

Vivado, BCD, Gray Code, Johnson, Fibonacci, Zedboard, Binary Counter, Ring Counter, VHDL

1.1 INTRODUCTION

The purpose of this experiment is to become familiar with the Vivado IDE and Zedboard. The experiment consists of building an LED circuit controlled by 6 different counters. Fig. 1.1 shows the top level block diagram for the counter design. The 6 different counters include Binary, Gray Code, BCD, Ring, Johnson and Fibonacci counters. Each counter is connected to a multiplexer that is passed through to the LEDs. In order to visualize the physical changes in the 8 LEDs, a frequency divider is used to divide the onboard clock of 100 MHz.

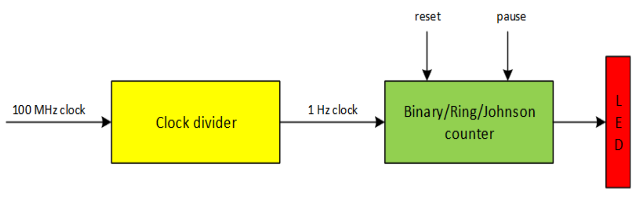
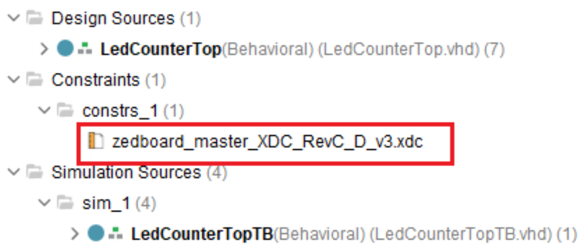


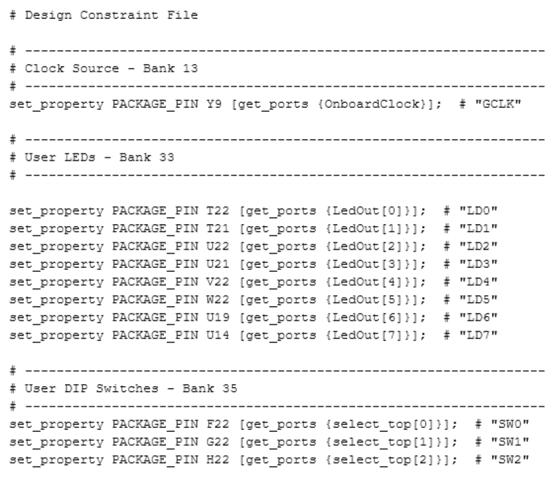
Fig. 1.1 General counter diagram with clock divider

In order to access the Onboard clock, switches, and LED pins, a constraint file must be included with the project. The constraint file for the Zedboard was downloaded from the Zedboard documentation webpage and imported to the project. In the XDC file, the pins and interfaces are directly mapped to the ports of the modules defined in the project. Fig 1.2 shows the Zedboard constraint file and an example of how the LED ports should be connected to the top module ports can be seen in Fig 1.3. The constraint file is included in the Appendix, but since the Zedboard is unavailable, the constraint file is not required for the simulation.



```
v Design Sources (1)
    > LedCounterTop(Behavioral) (LedCounterTop.vhd) (7)
v Constraints (1)
    v constrs_1 (1)
        zedboard_master_XDC_RevC_D_v3.xdc
v Simulation Sources (4)
    v sim_1 (4)
        > LedCounterTopTB(Behavioral) (LedCounterTopTB.vhd) (1)
```

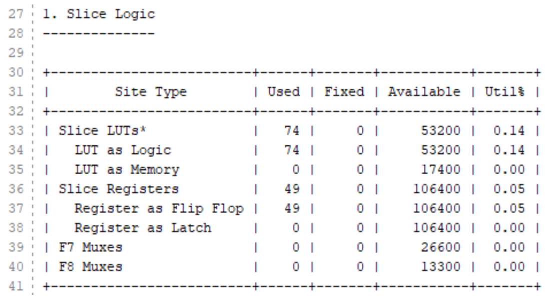
Fig. 1.2 The zedboard constraint file shown within the project hierarchy



```
# Design Constraint File

# ----------------------------------------------------------------------------
# Clock Source - Bank 13
# ----------------------------------------------------------------------------
set_property PACKAGE_PIN Y9 [get_ports {OnboardClock}];  # "GCLK"

# ----------------------------------------------------------------------------
# User LEDs - Bank 33
# ----------------------------------------------------------------------------

set_property PACKAGE_PIN T22 [get_ports {LedOut[0]}];  # "LD0"
set_property PACKAGE_PIN T21 [get_ports {LedOut[1]}];  # "LD1"
set_property PACKAGE_PIN U22 [get_ports {LedOut[2]}];  # "LD2"
set_property PACKAGE_PIN U21 [get_ports {LedOut[3]}];  # "LD3"
set_property PACKAGE_PIN V22 [get_ports {LedOut[4]}];  # "LD4"
set_property PACKAGE_PIN W22 [get_ports {LedOut[5]}];  # "LD5"
set_property PACKAGE_PIN U19 [get_ports {LedOut[6]}];  # "LD6"
set_property PACKAGE_PIN U14 [get_ports {LedOut[7]}];  # "LD7"

# ----------------------------------------------------------------------------
# User DIP Switches - Bank 35
# ----------------------------------------------------------------------------
set_property PACKAGE_PIN F22 [get_ports {select_top[0]}];  # "SW0"
set_property PACKAGE_PIN G22 [get_ports {select_top[1]}];  # "SW1"
set_property PACKAGE_PIN H22 [get_ports {select_top[2]}];  # "SW2"
```

Fig 1.3 The constraint file code for access to the LEDs, Onboard clock, and switches.

To find information about the designs utilization of FPGA resources, the design is synthesized. After the synthesis is done, a synthesis report file is created. The slice logic section within the report is shown in Fig 1.4



```
1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   74 |     0 |     53200 |  0.14 |
|   LUT as Logic          |   74 |     0 |     53200 |  0.14 |
|   LUT as Memory         |    0 |     0 |     17400 |  0.00 |
| Slice Registers         |   49 |     0 |    106400 |  0.05 |
|   Register as Flip Flop |   49 |     0 |    106400 |  0.05 |
|   Register as Latch     |    0 |     0 |    106400 |  0.00 |
| F7 Muxes                |    0 |     0 |     26600 |  0.00 |
| F8 Muxes                |    0 |     0 |     13300 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

Fig. 1.4 LUT and Flip Flop utilization details in the design synthesis report

In order to program the FPGA, the design must be implemented and then a bitstream must be generated. The bitstream file is located in the project folder, usually within the "/projectName.runs/impl_1" folder.

1.2 PROCEDURE AND RESULTS

The input clock frequency is 100 MHz. A clock frequency of 1 Hz is required to see the physical light changes in the LED. The following (1) shows the required number of bits for the frequency divider in order to achieve 1 Hz. A frequency of 1 Hz drastically increases the simulation time required, therefore, for the sake of simulation, the frequency divider was reduced to 1 bit and a frequency of 50 MHZ was used for the final simulations. Fig 1.5 shows the results of the frequency divider testbench that was run for 1.4 s. Looking closely, a pulse can be seen at around 1.34 s. This is the expected period for a frequency of 0.745 Hz.

$$\frac{100 * 1000 * 1000\ Hz}{2^{27}} = 0.74505 \text{ Hz} \qquad (1)$$

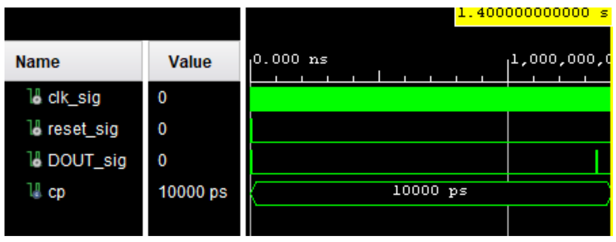


Fig. 1.5 The clock signal of about 1 Hz in simulation

6 different counters were designed in this experiment. This includes a binary, bcd, gray counter, johnson, ring, and a fibonacci counter. Fig 1.6 shows the simulation results of all 6 counters. Each result shows the expected sequence when for each type of counter. In Fig 1.6, the counters all pause when the pause signal is set to '1'. In Fig 1.7, the counters begin their sequences in reverse as the direction signal is set to '0'.

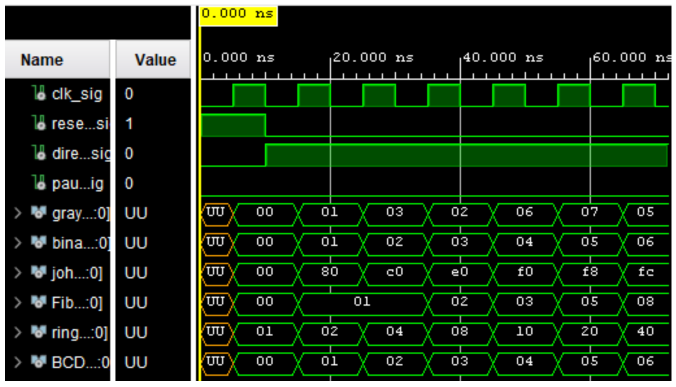


Fig 1.6 Simulation results of all 6 counters counting upwards

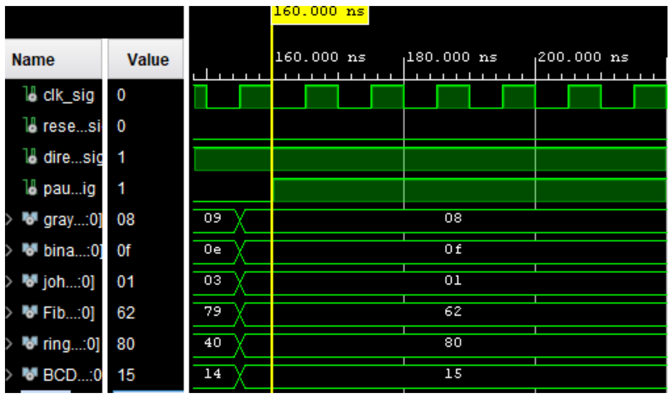


Fig 1.7 Simulation results of all 6 counters during a pause

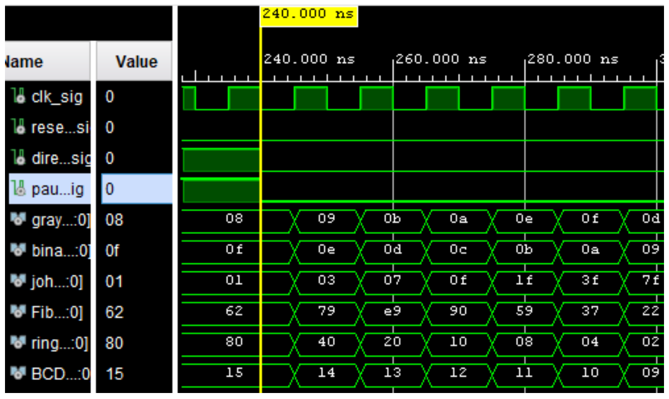


Fig 1.8 Simulation results of all 6 counters counting down

The last component of the overall design was the multiplexer. The multiplexer has a select line of 3 bits, each combination representing a type of counter. Fig 1.9 shows the simulation results of the multiplexer. For the sake of simplicity in testing, the types of counters are represented by one-hot code rather than an actual implementation of the counters. For example, binary is represented as “00000001” , a ring counter is represented as “00000010”, and johnson counter is represented as “00000100” and so on.

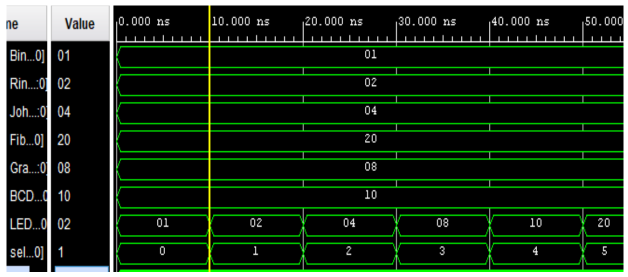


Fig 1.9 Simulation results of the multiplexer design

Fig 1.10 through Fig. 1.12 show the simulation results for the binary counter in the top level design with select set to 0. After the binary counter, the simulation continues from here to show each counter with the same inputs. Each different select showed the same pattern of counting up within their respective sequences and then counting backwards. While not exhaustive ,this test bench was a good indication that the design and implementation was correct.

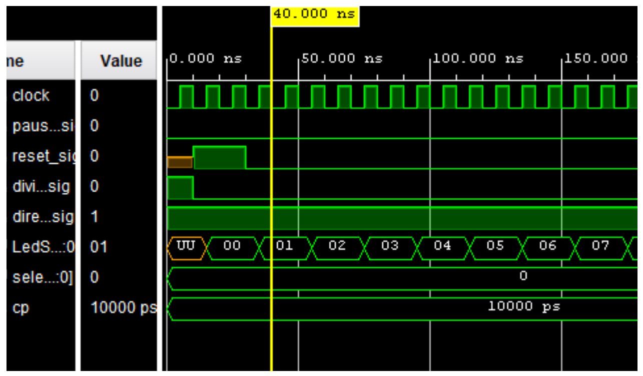


Fig 1.10 Simulation of the top level counter design for binary counting up

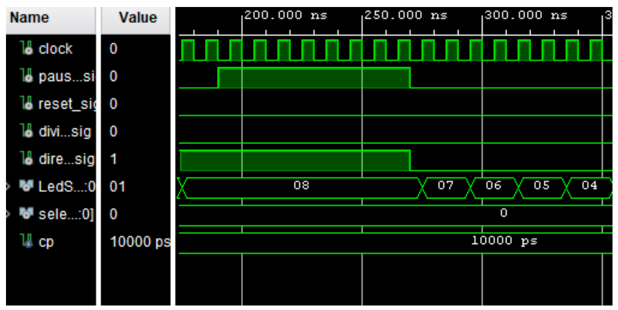


Fig 1.11 Simulation of the top level counter design for binary during pause

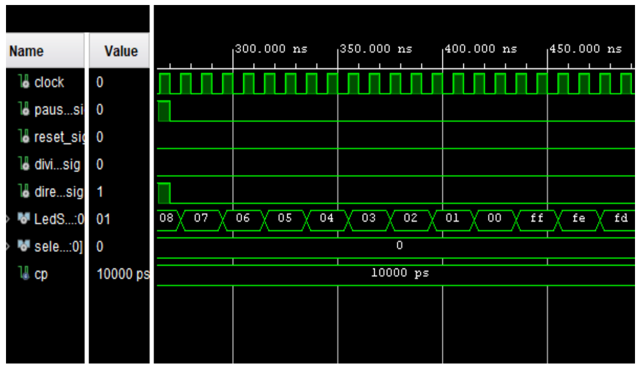


Fig 1.12 Simulation of the top level counter design for binary counting down

1.3. DISCUSSION AND CONCLUSION:

Overall the experiment was a success in implementation of this top level counter onto the Zedboard. The simulation results show the correct sequences for all 6 counters in our top level design. Unfortunately, the Zedboards are not available this semester for physical testing and implementation of the VHDL code; However, the file configurations for the Zedboard were properly set up, and the project was able to be synthesized and implemented.

REFERENCES

[1]http://www.ecs.csun.edu/~smirzaei/docs/ece524/ece524fall20-lab1-exploring-vivado-ide.pdf

[2]http://zedboard.org/sites/default/files/documentations/GS-AES-Z7EV-7Z020-G-V7-1.pdf

[3]https://www.ics.uci.edu/~jmoorkan/vhdlref/Synario%20VHDL%20Manual.pd

```vhdl
-- LedCounterTop.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity LedCounterTop is
    Port(
        OnboardClock, pause_top, reset_top,divider_reset, direction_top: in std_logic;
        select_top: in std_logic_vector(2 downto 0);
        LedOut: out std_logic_vector (7 downto 0)
    );
end LedCounterTop;

architecture Behavioral of LedCounterTop is
component FrequencyDivider is
    port(
        clk,reset:in std_logic;
        DOUT: out std_logic
    );
end component;

component GrayCounter is
    port(
        gray_clk,gray_reset,gray_direction,gray_pause:in std_logic;
        gray_DOUT: out std_logic_vector(7 downto 0);
        binary_DOUT: out std_logic_vector(7 downto 0)
    );
end component;

component RingCounter is
    port(
        clk,reset,direction,pause:in std_logic;
        DOUT: out std_logic_vector(7 downto 0)
    );
end component;

component JohnsonCounter is
    port(
        clk,reset,direction,pause:in std_logic;
        DOUT: out std_logic_vector(7 downto 0)
    );
end component;

component BCDCounter is
    port(
        clk,reset,direction,pause:in std_logic;
        DOUT: out std_logic_vector(7 downto 0)
    );
-- LedCounterTop.vhd Continued
end component;
```

```
component FibCounter is
    port(
        clk,reset,direction,pause:in std_logic;
        DOUT: out std_logic_vector(7 downto 0)
    );
end component;

component CounterMultiplexer is
    Port (
        counter_select: in std_logic_vector(2 downto 0);
        FibIn,GrayIn,BinIn,BCDIn,JohnIn,RingIn: in std_logic_vector(7 downto 0);
        LED: out std_logic_vector(7 downto 0)
     );
end component;

signal oneHz: std_logic;
signal BinMuxIn,RingMuxIn,JohnMuxIn: std_logic_vector(7 downto 0);
signal GrayMuxIn,BCDMuxIn,FibMuxIn : std_logic_vector(7 downto 0);
signal pause_sig,reset_sig,direction_sig: std_logic;
signal select_sig: std_logic_vector(2 downto 0);
signal OutSig: std_logic_vector(7 downto 0);

begin
    pause_sig <= pause_top;
    reset_sig <= reset_top;
    direction_sig <= direction_top;
    select_sig <= select_top;
    FreqDiv: FrequencyDivider
    port map(
        clk => OnboardClock,
        reset => divider_reset,
        DOUT => oneHz
    );

    BinGrayCntr: GrayCounter
    port map(
        gray_clk => oneHz, gray_reset => reset_sig,
        gray_direction => direction_sig, gray_pause => pause_sig,
        gray_DOUT => GrayMuxIn, binary_DOUT => BinMuxIn
    );
    RingCntr: RingCounter
    port map(
        clk => oneHz,reset => reset_sig,
        direction => direction_sig, pause => pause_sig,
     -- LedCounterTop.vhd Continued
       DOUT => RingMuxIn
    );
    JohnCntr: JohnsonCounter
    port map(
        clk => oneHz,reset => reset_sig,
        direction => direction_sig, pause => pause_sig,
        DOUT => JohnMuxIn
    );

    BCDCntr: BCDCounter
    port map(
        clk => oneHz,reset => reset_sig,
        direction => direction_sig, pause => pause_sig,
        DOUT => BCDMuxIn
    );

    FibCntr: FibCounter
    port map(
        clk => oneHz,reset => reset_sig,
        direction => direction_sig, pause => pause_sig,
        DOUT => FibMuxIn
    );
    CntrMux: CounterMultiplexer
    port map(
        counter_select => select_sig,
        FibIn => FibMuxIn,GrayIn => GrayMuxIn,
        BinIn => BinMuxIn ,BCDIn => BCDMuxIn,
        JohnIn => JohnMuxIn,RingIn => RingMuxIn,
        LED => OutSig
    );
    LedOut <= OutSig;
end Behavioral;
```

```
-- LedCounterTopTB.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity LedCounterTopTB is
--  Port ( );
end LedCounterTopTB;
architecture Behavioral of LedCounterTopTB is
component LedCounterTop is
       Port(
       OnboardClock, pause_top, reset_top,divider_reset, direction_top: in std_logic;
       select_top: in std_logic_vector(2 downto 0);
       LedOut: out std_logic_vector (7 downto 0)
       );
end component;
signal clock: std_logic := '0';
signal pause_sig,reset_sig,divider_reset_sig,direction_sig: std_logic;
signal LedSig: std_logic_vector(7 downto 0);
signal select_count: unsigned (2 downto 0);
constant cp: time := 10 ns;
begin
       DUT: LedCounterTop
       port map(
       OnboardClock => clock,
       pause_top => pause_sig,
       reset_top => reset_sig,
       divider_reset => divider_reset_sig,
       direction_top => direction_sig,
       select_top => std_logic_vector(select_count),
       LedOut => LedSig
       );
       clock_gen: process(clock)
       begin
       clock <= not clock after cp/2;
       end process;
       stim: process
       begin
       divider_reset_sig <= '1';
       select_count <= "000";
       pause_sig <= '0';
       direction_sig <= '1';
       wait for cp;
       divider_reset_sig <= '0';
       -- 000 Binary
       -- 001 Ring
       -- 010 John
       -- 011 Gray
-- LedCounterTopTB.vhd
       -- 100 BCD
       -- 101+ Fib
```

```
        for ii in 0 to 5 loop
                reset_sig <= '1';
                wait for 2*cp;
                reset_sig <= '0';
                direction_sig <= '1';
                wait for 16*cp;
                pause_sig <= '1';
                wait for 8*cp;
                pause_sig <= '0';
                direction_sig <= '0';
                wait for 24*cp;
                select_count <= select_count + 1;
        end loop;
        wait;
        end process;
end Behavioral;
```

```
--FrequencyDivider.vhd
library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;
use ieee.std_logic_misc.all;
entity FrequencyDivider is
	port(
	clk,reset:in std_logic;
	DOUT: out std_logic
	);
end FrequencyDivider;
architecture behavioral of FrequencyDivider is
signal count: unsigned(0 downto 0);
begin
	process(clk)
	begin
	if clk = '1' and clk' event then
		if reset = '1' then
		count <= (others => '0');
		else
		count <= count + 1;
		end if;
	end if;
	end process;
	DOUT <= and_reduce(std_logic_vector(count));
end architecture;
```

```
-- FrequencyDividerTB.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity FrequencyDividerTB is
--  Port ( );
end FrequencyDividerTB;

architecture Behavioral of FrequencyDividerTB is
component FrequencyDivider is
       port(
       clk,reset:in std_logic;
       DOUT: out std_logic
       );
end component;
signal clk_sig, reset_sig: std_logic := '0';
signal DOUT_sig: std_logic;
constant cp: time := 10 ns;
begin
       DUT: FrequencyDivider
       port map(
       clk => clk_sig,
       reset => reset_sig,
       DOUT => DOUT_sig
       );
       clock_gen:process(clk_sig)
       begin
       clk_sig <= not clk_sig after cp/2;
       end process;

       stim: process
       begin
       reset_sig <= '1';
       wait for cp;
       reset_sig <= '0';
       wait;
       end process;
end Behavioral;
```

```
--GrayBinaryCounter.vhd;
library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;
entity GrayCounter is
	port(
	gray_clk,gray_reset,gray_direction,gray_pause:in std_logic;
	gray_DOUT: out std_logic_vector(7 downto 0);
	binary_DOUT: out std_logic_vector(7 downto 0)
	);
end GrayCounter;

architecture behavioral of GrayCounter is
component BinaryCounter is
	port(
	clk,reset,direction,pause:in std_logic;
	DOUT: out std_logic_vector(7 downto 0)
	);
end component;
signal BinOut: std_logic_vector(7 downto 0);
begin
	BinCntr: BinaryCounter
	port map(clk => gray_clk,reset => gray_reset,
	direction => gray_direction,
	pause => gray_pause,
	DOUT => BinOut
	);
	binary_DOUT <= BinOut;
	gray_DOUT(7) <= BinOut(7);
	gray_DOUT(6) <= BinOut(7) xor BinOut(6);
	gray_DOUT(5) <= BinOut(6) xor BinOut(5);
	gray_DOUT(4) <= BinOut(5) xor BinOut(4);
	gray_DOUT(3) <= BinOut(4) xor BinOut(3);
	gray_DOUT(2) <= BinOut(3) xor BinOut(2);
	gray_DOUT(1) <= BinOut(2) xor BinOut(1);
	gray_DOUT(0) <= BinOut(1) xor BinOut(0);
end architecture;
```

```
--RingCounter.vhd;
library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;
entity RingCounter is
	port(
	clk,reset,direction,pause:in std_logic;
	DOUT: out std_logic_vector(7 downto 0)
	);
end RingCounter;

architecture behavioral of RingCounter is
signal count: unsigned (7 downto 0);
begin
	DOUT <= std_logic_vector(count);
	process(clk)
	begin
	if clk = '1' and clk' event then
		if reset = '1' then
		count <= "00000001";
		elsif pause = '1' then
		count <= count;
		elsif direction = '0' then
		count <=  count(0) & count(7 downto 1);
		else
		count <= count(6 downto 0) & count(7);
		end if;
	end if;
	end process;
end architecture;
```

```
--JohnsonCounter.vhd;
library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;
entity JohnsonCounter is
	port(
	clk,reset,direction,pause:in std_logic;
	DOUT: out std_logic_vector(7 downto 0)
	);
end JohnsonCounter;

architecture behavioral of JohnsonCounter is
signal count: unsigned (7 downto 0);
begin
	DOUT <= std_logic_vector(count);
	process(clk)
	begin
	if clk = '1' and clk' event then
		if reset = '1' then
		count <= (others => '0');
		elsif pause = '1' then
		count <= count;
		elsif direction = '1' then
		count <= (not count(0)) & count(7 downto 1);
		else
		count <= count(6 downto 0) & (not count(7));
		end if;
	end if;
	end process;
end architecture;
```

```
--BCDCounter.vhd;
library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;
entity BCDCounter is
       port(
       clk,reset,direction,pause:in std_logic;
       DOUT: out std_logic_vector(7 downto 0)
       );
end BCDCounter;
architecture behavioral of BCDCounter is
signal digit1,digit2: unsigned (3 downto 0);
begin
       process(clk)
       begin
       if clk = '1' and clk' event then
              if reset = '1' then
              digit1 <= (others => '0');
              digit2 <= (others => '0');
              elsif pause = '1' then
              digit1 <= digit1;
              digit2 <= digit2;
              elsif direction = '1' then
              if(digit1 = "1001" and digit2 = "1001")then
                     digit1 <= (others => '0');
                     digit2 <= (others => '0');
              elsif(digit1 = "1001") then
                     digit1 <= (others => '0');
                     digit2 <= digit2 + 1;
              else
                     digit1 <= digit1 + 1;
              end if;
              else
              if(digit1 = "0000" and digit2 = "0000")then
                     digit1 <= ("1001");
                     digit2 <= ("1001");
              elsif(digit1 = "0000") then
                     digit1 <= ("1001");
                     digit2 <= digit2 - 1;
              else
                     digit1 <= digit1 - 1;
              end if;
              end if;
       end if;
       end process;
       DOUT <= std_logic_vector(digit2&digit1);
end architecture;
```

```
--FibCounter.vhd;
library ieee;
use ieee.numeric_std.all;
use ieee.std_logic_1164.all;
entity FibCounter is
       port(
       clk,reset,direction,pause:in std_logic;
       DOUT: out std_logic_vector(7 downto 0)
       );
end FibCounter;
architecture behavioral of FibCounter is
signal count,temp: unsigned (7 downto 0);
begin
       DOUT <= std_logic_vector(temp);
       process(clk)
       begin
       if clk = '1' and clk' event then
              if reset = '1' then
              temp <= (others => '0');
              count <= "00000001";
              elsif pause = '1' then
              count <= count;
              temp <= temp;
              elsif direction = '1' then
              temp <= count;
              count <= count + temp;
              else
              count <= temp;
              temp <= count - temp;
              end if;
       end if;
       end process;
end architecture;
```

```vhdl
--CounterTB.vhd;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity CounterTB is
--  Port ( );
end CounterTB;
architecture Behavioral of CounterTB is
component GrayCounter is
	port(
	gray_clk,gray_reset,gray_direction,gray_pause:in std_logic;
	gray_DOUT: out std_logic_vector(7 downto 0);
	binary_DOUT: out std_logic_vector(7 downto 0)
	);
end component;
component JohnsonCounter is
	port(
	clk,reset,direction,pause:in std_logic;
	DOUT: out std_logic_vector(7 downto 0)
	);
end component;
component FibCounter is
	port(
	clk,reset,direction,pause:in std_logic;
	DOUT: out std_logic_vector(7 downto 0)
	);
end component;
component RingCounter is
	port(
	clk,reset,direction,pause:in std_logic;
	DOUT: out std_logic_vector(7 downto 0)
	);
end component;
component BCDCounter is
	port(
	clk,reset,direction,pause:in std_logic;
	DOUT: out std_logic_vector(7 downto 0)
	);
end component;
signal clk_sig,reset_sig,direction_sig,pause_sig: std_logic := '0';
signal gray_out_sig,binary_out_sig: std_logic_vector(7 downto 0);
signal johnson_out_sig,Fib_out_sig: std_logic_vector(7 downto 0);
signal ring_out_sig,BCD_out_sig: std_logic_vector(7 downto 0);
constant cp: time := 10 ns;
begin
	DUT0: GrayCounter
	port map(
	gray_clk => clk_sig, gray_reset => reset_sig,
	gray_direction => direction_sig,gray_pause => pause_sig,
	gray_DOUT => gray_out_sig,
	binary_DOUT => binary_out_sig
```

```
       );
       DUT1: JohnsonCounter
       port map(
       clk => clk_sig, reset => reset_sig,
       direction => direction_sig,pause => pause_sig,
       DOUT => johnson_out_sig
       );
       DUT2: FibCounter
       port map(
       clk => clk_sig, reset => reset_sig,
       direction => direction_sig,pause => pause_sig,
       DOUT => fib_out_sig
       );
       DUT3: RingCounter
       port map(
       clk => clk_sig, reset => reset_sig,
       direction => direction_sig,pause => pause_sig,
       DOUT => ring_out_sig
       );
       DUT4: BCDCounter
       port map(
       clk => clk_sig, reset => reset_sig,
       direction => direction_sig,pause => pause_sig,
       DOUT => BCD_out_sig
       );
       clock_gen:process(clk_sig)
       begin
       clk_sig <= not clk_sig after cp/2;
       end process;
       stim: process
       begin
       -- pause_sig <= '0';
       -- direction_sig <= '1';
       reset_sig <= '1';
       wait for cp;
       reset_sig <= '0';
       direction_sig <= '1';
       wait for 15*cp;
       pause_sig <= '1';
       wait for 8*cp;
       pause_sig <= '0';
       direction_sig <= '0';
       wait for 15*cp;
       wait;
       end process;
end Behavioral;
```

```vhdl
--CounterMultiplexer.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity CounterMultiplexer is
       Port (
       counter_select: in std_logic_vector(2 downto 0);
       FibIn,GrayIn,BinIn,BCDIn,JohnIn,RingIn: in std_logic_vector(7 downto 0);
       LED: out std_logic_vector(7 downto 0)
       );
end CounterMultiplexer;
architecture Behavioral of CounterMultiplexer is
begin
process(counter_select,BinIn,RingIn,JohnIn,GrayIn,BCDIn,FibIn)
begin
       case counter_select is
       when "000" => LED <= BinIn;
       when "001" => LED <= RingIn;
       when "010" => LED <= JohnIn;
       when "011" => LED <= GrayIn;
       when "100" => LED <= BCDIn;
       when others => LED <= FibIn;
       end case;
end process;
end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity CounterMultiplexerTB is
--  Port ( );
end CounterMultiplexerTB;
architecture Behavioral of CounterMultiplexerTB is
component CounterMultiplexer is
       Port (
       counter_select: in std_logic_vector(2 downto 0);
       FibIn, GrayIn, BinIn,BCDIn,JohnIn,RingIn: in std_logic_vector(7 downto 0);
       LED: out std_logic_vector(7 downto 0)
       );
end component;
signal FibSig,GraySig,BinSig,BCDSig,JohnSig,RingSig: std_logic_vector(7 downto 0);
signal LEDSig: std_logic_vector(7 downto 0);
signal selCnt: unsigned (2 downto 0);
constant cp: time := 10 ns;
begin
       DUT: CounterMultiplexer
       port map(
       counter_select => std_logic_vector(selCnt),
       FibIn => FibSig,
       GrayIn => GraySig,
       BinIn => BinSig,
       BCDIn => BCDSig,
       JohnIn => JohnSig,
       RingIn => RingSig,
       LED => LEDSig
       );
       STIM: process
       begin
       BinSig <= "00000001";
       RingSig <= "00000010";
       JohnSig <= "00000100";
       GraySig <= "00001000";
       BCDSig <= "00010000";
       FibSig <= "00100000";
       -- Rather than attaching the counter modules to each signal
       -- This test bench represents each counter by a one hot value
       selCnt <= "000";
       for ii in 0 to 7 loop
              wait for cp;
              selCnt <= selCnt + 1;
       end loop;
       wait;
       end process;
end Behavioral;
```

```
# Design Constraint File

# ----------------------------------------------------------------------------
# Clock Source - Bank 13
# ----------------------------------------------------------------------------
set_property PACKAGE_PIN Y9 [get_ports {OnboardClock}];  # "GCLK"

# ----------------------------------------------------------------------------
# User LEDs - Bank 33
# ----------------------------------------------------------------------------

set_property PACKAGE_PIN T22 [get_ports {LedOut[0]}];  # "LD0"
set_property PACKAGE_PIN T21 [get_ports {LedOut[1]}];  # "LD1"
set_property PACKAGE_PIN U22 [get_ports {LedOut[2]}];  # "LD2"
set_property PACKAGE_PIN U21 [get_ports {LedOut[3]}];  # "LD3"
set_property PACKAGE_PIN V22 [get_ports {LedOut[4]}];  # "LD4"
set_property PACKAGE_PIN W22 [get_ports {LedOut[5]}];  # "LD5"
set_property PACKAGE_PIN U19 [get_ports {LedOut[6]}];  # "LD6"
set_property PACKAGE_PIN U14 [get_ports {LedOut[7]}];  # "LD7"

# ----------------------------------------------------------------------------
# User DIP Switches - Bank 35
# ----------------------------------------------------------------------------
set_property PACKAGE_PIN F22 [get_ports {select_top[0]}];  # "SW0"
set_property PACKAGE_PIN G22 [get_ports {select_top[1]}];  # "SW1"
set_property PACKAGE_PIN H22 [get_ports {select_top[2]}];  # "SW2"
```

```
Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.
--------------------------------------------------------------------------------
--------------------------------------
| Tool Version : Vivado v.2020.1 (win64) Build 2902540 Wed May 27 19:54:49 MDT
2020
| Date         : Tue Sep 15 21:33:54 2020
| Host         : Ridge-PC running 64-bit major release  (build 9200)
| Command      : report_utilization -file LedCounterTop_utilization_synth.rpt -pb
LedCounterTop_utilization_synth.pb
| Design       : LedCounterTop
| Device       : 7z020clg484-1
| Design State : Synthesized
--------------------------------------------------------------------------------
--------------------------------------

Utilization Design Information

Table of Contents
-----------------
1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   74 |     0 |     53200 |  0.14 |
|   LUT as Logic          |   74 |     0 |     53200 |  0.14 |
|   LUT as Memory         |    0 |     0 |     17400 |  0.00 |
| Slice Registers         |   49 |     0 |    106400 |  0.05 |
|   Register as Flip Flop |   49 |     0 |    106400 |  0.05 |
|   Register as Latch     |    0 |     0 |    106400 |  0.00 |
| F7 Muxes                |    0 |     0 |     26600 |  0.00 |
| F8 Muxes                |    0 |     0 |     13300 |  0.00 |
+-------------------------+------+-------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and full
implementation, is typically lower. Run opt_design after synthesis, if not
already completed, for a more realistic count.
```

```
1.1 Summary of Registers by Type
--------------------------------

+--------+--------------+-------------+--------------+
| Total  | Clock Enable | Synchronous | Asynchronous |
+--------+--------------+-------------+--------------+
| 0    |            _ |          - |          - |
| 0    |            _ |          - |          Set |
| 0    |            _ |          - |          Reset |
| 0    |            _ |          Set |             - |
| 0    |            _ |   Reset |            - |
| 0    |            Yes |        - |          - |
| 0    |            Yes |        - |          Set |
| 0    |            Yes |        - |          Reset |
| 2    |            Yes |        Set |             - |
| 47   |            Yes |        Reset |            - |
+--------+--------------+-------------+--------------+


2. Memory
---------

+-----------------+------+-------+-----------+-------+
|     Site Type   | Used | Fixed | Available | Util% |
+-----------------+------+-------+-----------+-------+
| Block RAM Tile | 0 |    0 |    140 |  0.00 |
|   RAMB36/FIFO* | 0 |    0 |    140 |  0.00 |
|   RAMB18        |      0 |    0 |    280 |  0.00 |
+-----------------+------+-------+-----------+-------+
* Note: Each Block RAM Tile only has one FIFO logic available and therefore can
accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies
a Block RAM Tile, that tile can still accommodate a RAMB18E1


3. DSP
------

+-----------+------+-------+-----------+-------+
| Site Type | Used | Fixed | Available | Util% |
+-----------+------+-------+-----------+-------+
| DSPs      |      0 |    0 |    220 |  0.00 |
+-----------+------+-------+-----------+-------+


4. IO and GT Specific
---------------------
```

```
+------------------------------+------+-------+-----------+-------+
|          Site Type           | Used | Fixed | Available | Util% |
+------------------------------+------+-------+-----------+-------+
| Bonded IOB                   |   16 |     0 |   200 |  8.00 |
| Bonded IPADs                 |    0 |   0 |         2 |  0.00 |
| Bonded IOPADs                |    0 |   0 |   130 |  0.00 |
| PHY_CONTROL                  |    0 |   0 |         4 |  0.00 |
| PHASER_REF                   |    0 |   0 |         4 |  0.00 |
| OUT_FIFO                     |    0 |   0 |        16 |  0.00 |
| IN_FIFO                      |    0 |   0 |        16 |  0.00 |
| IDELAYCTRL                   |    0 |   0 |         4 |  0.00 |
| IBUFDS                       |    0 |   0 |   192 |  0.00 |
| PHASER_OUT/PHASER_OUT_PHY   |    0 |   0 |        16 |  0.00 |
| PHASER_IN/PHASER_IN_PHY      |    0 |   0 |        16 |  0.00 |
| IDELAYE2/IDELAYE2_FINEDELAY |    0 |   0 |   200 |  0.00 |
| ILOGIC                       |    0 |   0 |   200 |  0.00 |
| OLOGIC                       |    0 |   0 |   200 |  0.00 |
+------------------------------+------+-------+-----------+-------+


5. Clocking
-----------

+------------+------+-------+-----------+-------+
|  Site Type | Used | Fixed | Available | Util% |
+------------+------+-------+-----------+-------+
| BUFGCTRL   |    2 |   0 |        32 |  6.25 |
| BUFIO     |    0 |   0 |        16 |  0.00 |
| MMCME2_ADV |    0 |   0 |         4 |  0.00 |
| PLLE2_ADV  |    0 |   0 |         4 |  0.00 |
| BUFMRCE   |    0 |   0 |         8 |  0.00 |
| BUFHCE    |    0 |   0 |        72 |  0.00 |
| BUFR      |    0 |   0 |        16 |  0.00 |
+------------+------+-------+-----------+-------+


6. Specific Feature
-------------------

+-------------+------+-------+-----------+-------+
|  Site Type  | Used | Fixed | Available | Util% |
+-------------+------+-------+-----------+-------+
| BSCANE2    |    0 |   0 |         4 |  0.00 |
| CAPTUREE2   |   0 |   0 |         1 |  0.00 |
| DNA_PORT   |    0 |   0 |         1 |  0.00 |
| EFUSE_USR   |   0 |   0 |         1 |  0.00 |
| FRAME_ECCE2 |   0 |   0 |         1 |  0.00 |
| ICAPE2     |      0 |    0 |          2 |  0.00 |
| STARTUPE2   |    0 |    0 |          1 |  0.00 |
| XADC       |      0 |    0 |          1 |  0.00 |
+-------------+------+-------+-----------+-------+


7. Primitives
-------------

+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| FDRE       |   47 |     Flop & Latch |
| LUT3       |   34 |                LUT |
| LUT6       |   26 |                LUT |
| LUT2       |   15 |                LUT |
| OBUF       |     8 |                IO |
| IBUF       |     8 |                IO |
| CARRY4   | 6 |           CarryLogic |
| LUT5       |     3 |                LUT |
| LUT4       |     3 |                LUT |
| LUT1       |     3 |                LUT |
| FDSE       |     2 |          Flop & Latch |
| BUFG       |     2 |                Clock |
+----------+------+---------------------+


8. Black Boxes
--------------

+----------+------+
| Ref Name | Used |
+----------+------+


9. Instantiated Netlists
------------------------

+----------+------+
| Ref Name | Used |
+----------+------+
```