Universidad
LATINA de Panamá
SUMMUM DESIDERIUM SAPIENTIA

Trabajo presentado por:
Tsarev Alvarado
Jarod Varela

Cédula de identidad personal:
3-743-86
8-936-469

Asignatura:
Diseño de Sistemas Digitales

Tema:
Proyecto final - Basys Board Binary Count-To Game

Fecha de entrega:
26 de abril de 2019

Introducción

Este tutorial instructivo le muestra cómo construir un juego que multiplica la capacidad de un jugador para contar de 0 a 15 en binario utilizando los interruptores del tablero Digilent Basys 3. Mientras un jugador está cambiando a través de los números, la fila de LED en el tablero de Basys 3 se iluminará para mostrar el número en el que se encuentra el jugador actualmente. El tiempo se mostrará en la pantalla de siete segmentos de la placa y pasará de 0,00 segundos a 99,99 segundos. Si un jugador se queda sin tiempo, los LED mostrarán un estado de pérdida.

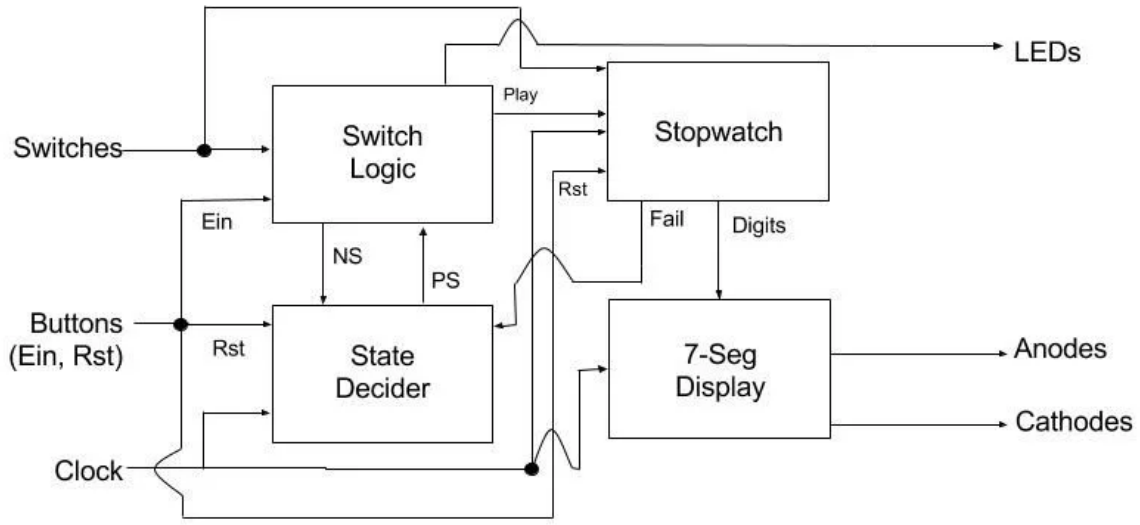
Para usar este código, necesitarás:

- Una placa Digilent Basys 3 FPGA

- Computadora con el software Xilinx Vivado (Webpack Edition funcionará). Para este módulo, utilizamos la versión 2018.3.

- Un cable micro USB (capaz de transferencia de datos)

Paso 1:
Cómo funciona



Entradas: el módulo utiliza los 4 conmutadores situados más a la derecha en la placa Basys. También hace uso de los pulsadores central e izquierdo ubicados cerca de la mitad de la placa, así como del reloj de 100 MHz incorporado.

Salidas: El módulo utiliza la pantalla de 4 dígitos de 7 segmentos en la placa Basys, que consiste internamente en 4 ánodos que controlan cada dígito, todos los cuales están conectados a través de cátodos comunes que se refieren a los segmentos. También hace uso de los LED de la placa Basys ubicada cerca de la parte inferior de la placa, sobre los interruptores.

Dentro del programa, hay múltiples "submódulos" que son funcionalmente diferentes entre sí pero que contribuyen al código completo en su totalidad, en forma de bloques de proceso:

Switch Logic: este bloque de proceso controla la lógica del switch para el módulo. Esto funciona en gran cooperación con el bloque de proceso de decisión estatal. La forma en que está diseñado este bloque, los interruptores mueven al usuario a través

de varios estados de estados únicos hasta que se transfieren al estado "ganador". Este bloque de proceso también controla la fila de LED que se iluminan cuando el jugador atraviesa los diferentes estados. Una vez que se haya inicializado el juego, el LED se encenderá en consecuencia para que el usuario sepa qué número debe contar hasta el siguiente, hasta que se muestren todos los LED.

Decisor de estado: este bloque de proceso indica el estado del número en el que se encuentra el jugador mientras atraviesan las combinaciones de números secuenciales en los interruptores. A medida que el jugador atraviesa los diferentes estados a través del bloque de proceso de Switch Logic, se encenderá un nuevo LED hasta que alcance el último, donde el reloj se detiene y se muestra un estado de ganancia. Alternativamente, si el reproductor se queda sin tiempo, se muestra un estado de falla.

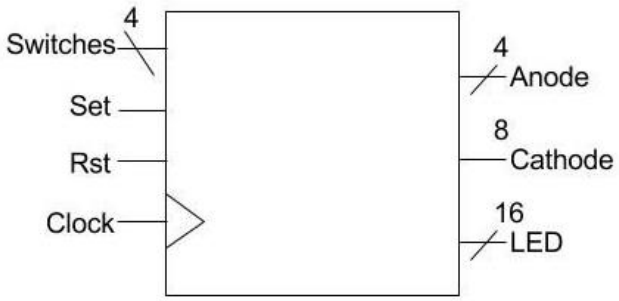
Cronómetro: este bloque de proceso controla los dígitos que se muestran en la pantalla de siete segmentos, para que puedan recorrer los números en orden e incrementar el siguiente dígito una vez que un dígito haya completado su ciclo 0-9. Si el cronómetro alcanza su valor máximo (en este módulo, 100 segundos), entonces retransmitirá una señal activa de "Fallo" al bloque de proceso de Decidir de Estado, que moverá el estado automáticamente al Estado de Fallo.

Siete segmentos: este bloque de proceso controla los LED reales que deben encenderse cuando el cronómetro está encendido, ya que los LED en la pantalla de siete segmentos comparten un ánodo y un cátodo comunes. Cada dígito dentro del Segmento 7 está vinculado a la señal dentro del módulo Cronómetro y tiene directivas específicas a seguir dependiendo de cuál sea el valor del dígito en un momento determinado.

Se adjunta un diagrama que muestra la arquitectura del circuito del módulo, que muestra las relaciones interconectadas entre los 4 bloques de proceso.

Paso 2:
Cableando la placa

Para cablear la placa, deberá crear un archivo de restricción y asegurarse de que la placa esté conectada a los componentes adecuados especificados en la placa. Para hacerlo, puede copiar y pegar el archivo Basys3_Master.xdc en su archivo de restricciones, y usar el Manual de referencia de Basys3 para asegurarse de que conecta la placa a los componentes correctos. Además, proporcionaremos nuestro propio archivo de restricciones para evitar la molestia.



Paso 3:
Programación del tablero Basys

Una vez que haya completado la configuración para el juego del contador, puede generar el flujo de bits para el código, que también ejecutará la síntesis y la implementación para el tablero. Durante esta etapa, pueden aparecer algunos errores, así que asegúrate de que se resuelvan antes de intentar programar el tablero para asegurarte de que el juego funcione correctamente. Una vez que se haya generado el flujo de bits, puede abrir el objetivo para el dispositivo y usar la función de conexión automática para programar la placa Basys3.

Se adjunta el código para el juego Count-To-15, que tiene un cronómetro de cronómetro máximo de 100 segundos, así como el código del Basys3_Master.xdc con sus respectivas modificaciones para este proyecto.

Código de Count-To-15

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

entity Game is
    Port ( Clock : in STD_LOGIC;
           Switch : in STD_LOGIC_VECTOR (3 downto 0);
           Ein : in STD_LOGIC;
           Rst : in STD_LOGIC;
           LED : out STD_LOGIC_Vector (15 downto 0);
           Anode : out STD_LOGIC_VECTOR (3 downto 0);
           Cathode : out STD_LOGIC_VECTOR (7 downto 0));
end Game;

architecture Behavioral of Game is

signal PS : Std_logic_vector (4 downto 0) := "00000"; -- signal for our FSM Present
State
signal NS : Std_logic_vector (4 downto 0) := "00000"; -- signal for our FSM Next
State
signal Play : std_logic := '0'; -- signal for our enable for the stopwatch logic
signal digit1 : integer := 0; -- signal for Digit 1 (Right Most)
signal digit2 : integer := 0; -- signal for Digit 2 (Second to Right)
signal digit3 : integer := 0; -- signal for Digit 3 (Second to Left)
signal digit4 : integer := 0; -- signal for Digit 4 (Left Most)
signal Fail : std_logic := '0'; -- signal for our Fail condition, once toggled
high, the fail sequence will initiate

begin

    State_Decider: process (Clock, Rst, Fail) -- Controls the States
    begin
        If Rst = '1' then -- Reset takes precedence, over anything else
            PS <= "00000";
        elsif Fail = '1' then -- Fail only goes live if the timer reaches 100
seconds
            PS <= "11011";
        elsif (Rising_edge(Clock)) then -- the Basys board has a 100MHz Clock,
which on every rising edge of the square wave pulse, we assign our stages
                PS <= NS;
        end if;
    end process;

    Switch_Logic: process (PS, Switch, Ein) -- Logic controlling the switches,
which is dependant on the stages. LED will light up, telling you what number to
count to next
    begin
        NS <= PS; --fixes a PS and NS latch
        Case (PS) is
            When "00000" => --Pre-set Stage
                LED <= "0000000000000000";
```

```
                Play <= '0'; -- Play must be defined in every case here, or else
creates a latch
                if Switch = "0000" and Ein = '1' then
                    NS <= "00001";
                end if;
            when "00001" => -- Set Stage
                LED <= "0000000000000001";
                Play <= '0';
                if Switch = "0001" then
                    NS <= "00010";
                end if;
            when "00010" => --Stage where someone is playing, switch is at 0001
                LED <= "0000000000000011";
                Play <= '1';
                if Switch = "0010" then
                    NS <= "00011";
                end if;
            when "00011" =>
                LED <= "0000000000000111";
                Play <= '1';
                if Switch = "0011" then
                    NS <= "00100";
                end if;
            when "00100" =>
                LED <= "0000000000001111";
                Play <= '1';
                if Switch = "0100" then
                    NS <= "00101";
                end if;
            when "00101" =>
                LED <= "0000000000011111";
                Play <= '1';
                if Switch = "0101" then
                    NS <= "00110";
                end if;
            when "00110" =>
                LED <= "0000000000111111";
                Play <= '1';
                if Switch = "0110" then
                    NS <= "00111";
                end if;
            when "00111" =>
                LED <= "0000000001111111";
                Play <= '1';
                if Switch = "0111" then
                    NS <= "01000";
                end if;
            when "01000" =>
                LED <= "0000000011111111";
                Play <= '1';
                if Switch = "1000" then
                    NS <= "01001";
                end if;
            when "01001" =>
                LED <= "0000000111111111";
```

```
                Play <= '1';
                if Switch = "1001" then
                    NS <= "01010";
                end if;
            when "01010" =>
                LED <= "0000001111111111";
                Play <= '1';
                if Switch = "1010" then
                    NS <= "01011";
                end if;
            when "01011" =>
                LED <= "0000011111111111";
                Play <= '1';
                if Switch = "1011" then
                    NS <= "01100";
                end if;
            when "01100" =>
                LED <= "0000111111111111";
                Play <= '1';
                if Switch = "1100" then
                    NS <= "01101";
                end if;
            when "01101" =>
                LED <= "0001111111111111";
                Play <= '1';
                if Switch = "1101" then
                    NS <= "01110";
                end if;
            when "01110" =>
                LED <= "0011111111111111";
                Play <= '1';
                if Switch = "1110" then
                    NS <= "01111";
                end if;
            when "01111" =>
                LED <= "0111111111111111";
                Play <= '1';
                if Switch = "1111" then
                    NS <= "10000";
                end if;
            when "10000" => -- Win Stage, time recorded on the 7 segment
                LED <= "1111111111111111";
                Play <= '0';
            when "11011" => -- Failed Stage, only occurs when timer reaches 100
seconds
                LED <= "1010101010101010";
                Play <= '0';
            when others => -- Failsafe
                LED <= "0000000000000000";
                Play <= '0';
        end case;
    end process;

    Stopwatch : process (Play, Rst, Clock)
        variable Count: integer := 0;
```

```
    begin
        if Rst = '1' then
            Count := 0;
            Digit1 <= 0; -- Controls the first digit of the seven segment
            Digit2 <= 0; -- Controls the second digit of the seven segment
            Digit3 <= 0; -- Controls the third digit of the seven segment
            Digit4 <= 0; -- Controls the fourth digit of the seven segment
            Fail <= '0'; -- Fail is our fail signal, which once triggered, will
go to the stage "11011"
        elsif Play = '0' then -- If Play is off, then count will be the held to
the value as it was
            Count := Count;
        elsif Play = '1' and rst = '0' then
            if (rising_edge(Clock)) then
                Count := Count + 1;
                if Count = 1000000 then -- Count will only increment Digit0 every
hundreth of a second
                    Digit1 <= Digit1 + 1;
                    Count := 0;
                    if Digit1 = 9 then
                        Digit2 <= Digit2 + 1;
                        Digit1 <= 0;
                        if Digit2 = 9 then
                            Digit3 <= Digit3 + 1;
                            Digit2 <= 0;
                            if Digit3 = 9 then
                                Digit4 <= Digit4 + 1;
                                Digit3 <= 0;
                                if Digit4 = 9 then -- Once the last digit rolls
over
                                    digit4 <= 0;
                                    Fail <= '1';
                                end if;
                            end if;
                        end if;
                    end if;
                end if;
            end if;
        end if;
    end process;

    Seven_Seg_Display : process (Clock, Digit1, Digit2, Digit3, Digit4) --
Controls the 7 Segment Display on the Basys Board
            variable counting : integer := 0;
        begin
            if (rising_edge(Clock)) then
                Counting := counting +1;
                If Counting = 1 then
                    Anode <= "1110"; --Utilize Right-Most Digit of 7-Segment Only
                    If Digit1 = 0 then --0
                        Cathode <= "00000011";
                    elsif Digit1 = 1 then --1
                        Cathode <= "10011111";
                    elsif Digit1 = 2 then --2
                        Cathode <= "00100101";
```

```
                    elsif Digit1 = 3 then --3
                        Cathode <= "00001101";
                    elsif Digit1 = 4 then --4
                        Cathode <= "10011001";
                    elsif Digit1 = 5 then --5
                        Cathode <= "01001001";
                    elsif Digit1 = 6 then --6
                        Cathode <= "01000001";
                    elsif Digit1 = 7 then --7
                        Cathode <= "00011111";
                    elsif Digit1 = 8 then --8
                        Cathode <= "00000001";
                    elsif Digit1 = 9 then--9
                        Cathode <= "00011001";
                    end if;
                elsif Counting = 400000 then -- Refreshes the Second Right-Most
Segment
                    Anode <= "1101";
                    If Digit2 = 0 then --0
                        Cathode <= "00000011";
                    elsif Digit2 = 1 then --1
                        Cathode <= "10011111";
                    elsif Digit2 = 2 then --2
                        Cathode <= "00100101";
                    elsif Digit2 = 3 then --3
                        Cathode <= "00001101";
                    elsif Digit2 = 4 then --4
                        Cathode <= "10011001";
                    elsif Digit2 = 5 then --5
                        Cathode <= "01001001";
                    elsif Digit2 = 6 then --6
                        Cathode <= "01000001";
                    elsif Digit2 = 7 then --7
                        Cathode <= "00011111";
                    elsif Digit2 = 8 then --8
                        Cathode <= "00000001";
                    elsif Digit2 = 9 then--9
                        Cathode <= "00011001";
                    end if;
                elsif Counting = 800000 then -- Refreshes the Second Left-Most
Segment
                    Anode <= "1011";
                    If Digit3 = 0 then --0
                        Cathode <= "00000010";
                    elsif Digit3 = 1 then --1
                        Cathode <= "10011110";
                    elsif Digit3 = 2 then --2
                        Cathode <= "00100100";
                    elsif Digit3 = 3 then --3
                        Cathode <= "00001100";
                    elsif Digit3 = 4 then --4
                        Cathode <= "10011000";
                    elsif Digit3 = 5 then --5
                        Cathode <= "01001000";
                    elsif Digit3 = 6 then --6
```

```
                    Cathode <= "01000000";
                elsif Digit3 = 7 then --7
                    Cathode <= "00011110";
                elsif Digit3 = 8 then --8
                    Cathode <= "00000000";
                elsif Digit3 = 9 then--9
                    Cathode <= "00011000";
                end if;
            elsif Counting = 1200000 then --Refreshes the Left-Most digit of
the 7 Segment
                Anode <= "0111";
                If Digit4 = 0 then --0
                    Cathode <= "00000011";
                elsif Digit4 = 1 then --1
                    Cathode <= "10011111";
                elsif Digit4 = 2 then --2
                    Cathode <= "00100101";
                elsif Digit4 = 3 then --3
                    Cathode <= "00001101";
                elsif Digit4 = 4 then --4
                    Cathode <= "10011001";
                elsif Digit4 = 5 then --5
                    Cathode <= "01001001";
                elsif Digit4 = 6 then --6
                    Cathode <= "01000001";
                elsif Digit4 = 7 then --7
                    Cathode <= "00011111";
                elsif Digit4 = 8 then --8
                    Cathode <= "00000001";
                elsif Digit4 = 9 then--9
                    Cathode <= "00011001";
                end if;
            elsif Counting = 1600000 then -- Resets Counting to 0, which
repeats this process
                Counting := 0;
            end if;
        end if;
    end process;
end Behavioral;
```

Modificaciones al Basys3_Master.xdc

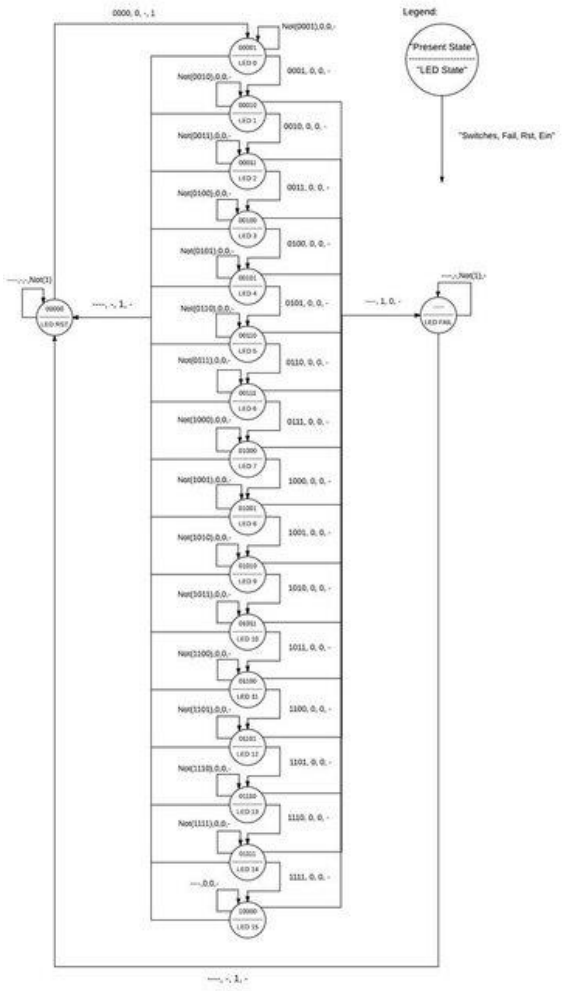
```
set_property PACKAGE_PIN W5 [get_ports {Clock}]
set_property IOSTANDARD LVCMOS33 [get_ports {Clock}]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports
{Clock}]

set_property PACKAGE_PIN L1 [get_ports {LED[15]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[15]}]

set_property PACKAGE_PIN P1 [get_ports {LED[14]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[14]}]

set_property PACKAGE_PIN N3 [get_ports {LED[13]}]
```

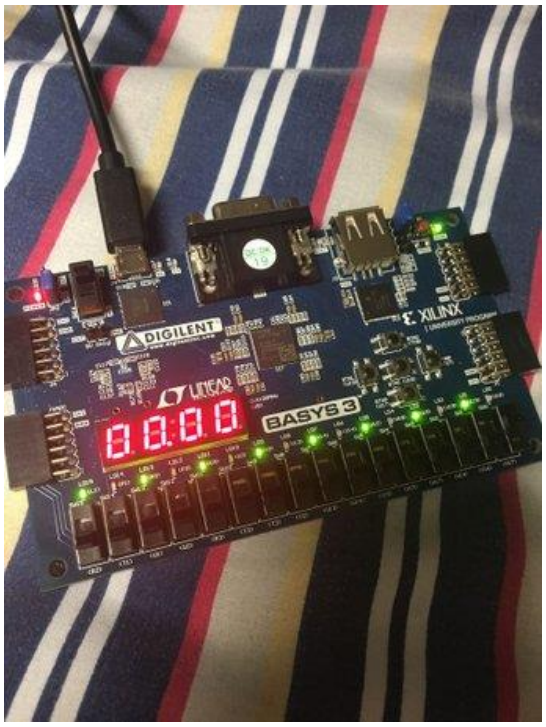
```
set_property IOSTANDARD LVCMOS33 [get_ports {LED[13]}]

set_property PACKAGE_PIN P3 [get_ports {LED[12]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[12]}]

set_property PACKAGE_PIN U3 [get_ports {LED[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[11]}]

set_property PACKAGE_PIN W3 [get_ports {LED[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[10]}]

set_property PACKAGE_PIN V3 [get_ports {LED[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[9]}]

set_property PACKAGE_PIN V13 [get_ports {LED[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[8]}]

set_property PACKAGE_PIN V14 [get_ports {LED[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[7]}]

set_property PACKAGE_PIN U14 [get_ports {LED[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[6]}]

set_property PACKAGE_PIN U15 [get_ports {LED[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[5]}]

set_property PACKAGE_PIN W18 [get_ports {LED[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[4]}]

set_property PACKAGE_PIN V19 [get_ports {LED[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[3]}]

set_property PACKAGE_PIN U19 [get_ports {LED[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[2]}]

set_property PACKAGE_PIN E19 [get_ports {LED[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[1]}]

set_property PACKAGE_PIN U16 [get_ports {LED[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[0]}]

set_property PACKAGE_PIN W17 [get_ports {Switch[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Switch[3]}]

set_property PACKAGE_PIN W16 [get_ports {Switch[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Switch[2]}]

set_property PACKAGE_PIN V16 [get_ports {Switch[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Switch[1]}]

set_property PACKAGE_PIN V17 [get_ports {Switch[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Switch[0]}]

set_property PACKAGE_PIN U18 [get_ports {Rst}]
set_property IOSTANDARD LVCMOS33 [get_ports {Rst}]
```

```
set_property PACKAGE_PIN W19 [get_ports {Ein}]
set_property IOSTANDARD LVCMOS33 [get_ports {Ein}]

set_property PACKAGE_PIN W4 [get_ports {Anode[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Anode[3]}]

set_property PACKAGE_PIN V4 [get_ports {Anode[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Anode[2]}]

set_property PACKAGE_PIN U4 [get_ports {Anode[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Anode[1]}]

set_property PACKAGE_PIN U2 [get_ports {Anode[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Anode[0]}]

set_property PACKAGE_PIN W7 [get_ports {Cathode[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathode[7]}]

set_property PACKAGE_PIN W6 [get_ports {Cathode[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathode[6]}]

set_property PACKAGE_PIN U8 [get_ports {Cathode[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathode[5]}]

set_property PACKAGE_PIN V8 [get_ports {Cathode[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathode[4]}]

set_property PACKAGE_PIN U5 [get_ports {Cathode[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathode[3]}]

set_property PACKAGE_PIN V5 [get_ports {Cathode[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathode[2]}]

set_property PACKAGE_PIN U7 [get_ports {Cathode[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathode[1]}]

set_property PACKAGE_PIN V7 [get_ports {Cathode[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Cathode[0]}]
```

Paso 4:
Cómo usarlo

LED RST	**“0000000000000000”**
LED 0	**“0000000000000001”**
LED 1	**“0000000000000011”**
LED 2	**“0000000000000111”**
LED 3	**“0000000000001111”**
LED 4	**“0000000000011111”**
LED 5	**“0000000000111111”**
LED 6	**“0000000001111111”**
LED 7	**“0000000011111111”**
LED 8	**“0000000111111111”**
LED 9	**“0000001111111111”**
LED 10	**“0000011111111111”**
LED 11	**“0000111111111111”**
LED 12	**“0001111111111111”**
LED 13	**“0011111111111111”**
LED 14	**“0111111111111111”**
LED 15	**“1111111111111111”**
LED Fail	**“1010101010101010”**





El juego se puede reiniciar en cualquier momento mediante el botón de reinicio programado (BTNC U18 en nuestro código), y esto borrará el reloj y cualquier LED que se haya encendido. Para poner el juego en su posición de inicio, el Interruptor 0 a 3 debe estar en la posición de apagado, antes de presionar el botón de ajuste (BTNC W19 en nuestro código) funcionará y encenderá el LED en la posición 0. Una vez que se active el interruptor 0 (LSB), el reloj se iniciará y continuará funcionando hasta que todas las etapas de progreso del juego se hayan completado en orden, o hasta que el cronómetro alcance su valor máximo, lo que enviará al usuario al estado de "Fallo" del juego.

Se adjunta el Diagrama de estado del bloque de proceso de lógica de conmutación, una tabla de luces LED que se

correlacionan con el diagrama de estado y una imagen del estado de falla.

Conclusión

El Basys 3 es una placa de desarrollo de FPGA de nivel de entrada diseñada exclusivamente para Vivado® Design Suite con la arquitectura Xilinx® Artix®-7-FPGA. Basys 3 es la incorporación más reciente a la popular línea Basys de tableros de desarrollo de FPGA para estudiantes o principiantes que recién comienzan con la tecnología FPGA. Basys 3 incluye las características estándar que se encuentran en todas las placas Basys: hardware completo listo para usar, una gran colección de dispositivos de E / S incorporados, todos los circuitos de soporte FPGA requeridos y una versión gratuita de herramientas de desarrollo y para un alumno -nivel de precio punto.