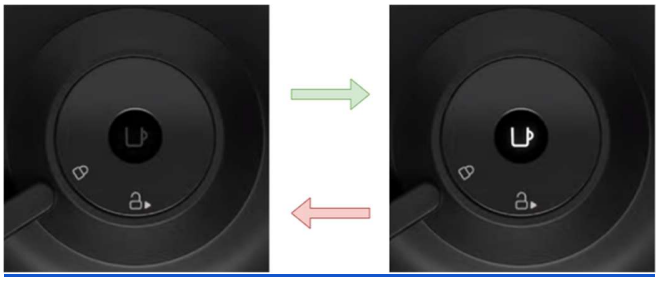
22.10.2024

Coffer Maker FPGA

LED Interface for a Capsule Coffee Maker

Task Description

A company building capsule coffee makers has asked for a circuit that can blink an indicator LED at specific frequencies depending on what size of coffee the customer wants:



The blinking frequencies are predetermined as follows:

- Espresso (40 ml) → 1 Hz
- Lungo (110 ml) → 2 Hz

The coffee size selection will be implemented with a single binary switch (0: espresso, 1: lungo) and there will be a second master power switch. The rest of the coffee machine is irrelevant, only the circuitry just described should be considered here. The company has agreed to an FPGA proof-of-concept implementation for now.

Moreover, to them there will be a child protection system to lock system without correct password. In that sense, specific three passwords will work with power switch so that coffee machine safely work.

Acceptance Test

The company representative will switch the power ON and simply count how many times the LED blinks within a 10-second interval with a stopwatch. Based on the provided blinking frequency requirements, the LED needs to blink 10 times within that 10-second interval for the espresso mode, and 20 times for the lungo mode.

Power switches only works when one of the passwords correctly entered. Also password switches should be open for user understand which switches open to enter correct password.

Pw#: password bit

Pw3|pw2|pw1

0 0 0

0 0 1

0 1 1

Requirements

The requirements of the system are listed below, including interface (**IRQ#**), functional (**FRQ#**) and timing (**TRQ#**) requirements.

Interface requirements

IRQ1 – Input: 2 independent binary switches (HIGH - LOW) that maintain their state once toggled by the user. The states of these switches are called "*PWR"* and "*SEL"* in the following.

IRQ2 – Output: illumination intensity of 1 LED. This LED is called "*LED"* in the following.

Functional requirements

FRQ1 – Read *PWR* and manipulate the state of an LED blinking routine with it. Specifically, the blinking routine should start when *PWR* = HIGH, and stop when *PWR* = LOW.

FRQ2 – The LED blinking routine should alternate the illumination level of *LED* between its highest and lowest possible levels at specified frequencies called "*FE*" and "*FL*" in the following.

FRQ3 – Read *SEL* to choose between *FE* and *FL* for the LED blinking routine. Specifically, *FL* should be chosen when *SEL* = HIGH and *FE* when *SEL* = LOW.

Timing requirements

TRQ1 – The LED blinking routine should have *FE* = 1 Hz and *FL* = 2 Hz with 50% duty cycle.

TRQ2 – Counter count at 49999998 for blink cycle.

Implementation

For using VHDL code vivado used and basys3 board helps to implement it. Clock is set to 10 period so 10 ns is the period of the basys3.

```
## - rename the used ports (in each line, after get_ports) according to the top level sign

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

I redesign switches such as:
-- sw0: for power on and off
-- sw1: for espresso on and lungo off
-- sw2: most significant bit pw3
-- sw3: second most pw2
-- sw4: least significant bit pw1

```
## Switches
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports {sw[0]}]
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports {sw[1]}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 } [get_ports {sw[2]}]
set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 } [get_ports {sw[3]}]
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 } [get_ports {sw[4]}]
```

Leds are designed for reflect which switches open, espresso or lungo and blinks for show hertz.
--led0: Blinks to show hertz of system.
--led1: Espresso or lungo led.
--led2: most significant bit led led.
--led3: second most significant bit led.
--led4: least significant bit led.

```
## LEDs
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports {led[0]}]
set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports {led[1]}]
set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports {led[2]}]
set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports {led[3]}]
set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports {led[4]}]
```

Creating a counter variable to 5 million and count each clock helps to reach frequency goal. However, lungo has more frequency so I count twice at each clock. That means it get 2 Hertz blink value. Without an external counter that gives more efficient solution to this problem.

I use 5 switches to represent inputs, that helps to get my input and process. Therefore I have 5 leds to indicate password (3 of leds),type of coffee(1 of leds), frequency of led(1 of the leds). Using std_logic_vector helps to more clear code. If statements helps me to decide power on and one of the password correct and then it check espresso or lungo and it blinks according to that values.

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

entity coffeemaker is
  Port ( clk : in  STD_LOGIC;
         led : out STD_LOGIC_Vector(4 downto 0) ;
         sw  : in STD_LOGIC_Vector (4 downto 0)
       );
end coffeemaker;

-- sw0 for power on and off
-- sw1 for espresso on and lungo off
-- sw2 most significant bit pw3
-- sw3 second most pw2
-- sw4 least significant bit pw1
-- passwords: "000" "001" "011"
architecture Behavioral of coffeemaker is
  signal pulse : std_logic := '0';
  signal count : integer range 0 to 99999999 := 0;
begin
  process(clk, sw)
  begin


    if sw(0) = '0' or not ((sw(2)= '0' and sw(3)= '0' and sw(4)= '0')
    or (sw(2)= '0' and sw(3)= '0' and sw(4)= '1')or (sw(2)= '0' and sw(3)= '1' and sw(4)= '1') ) then
        pulse <= '0';
    elsif clk'event and clk = '1' then
    --99999999
```

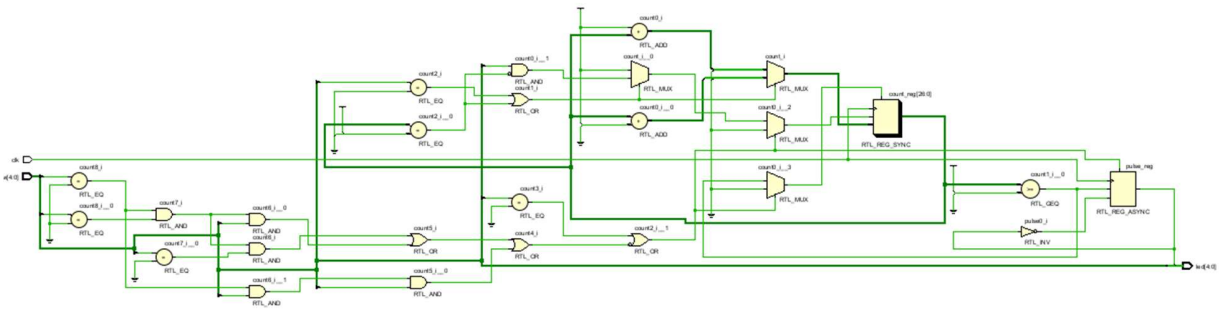

```vhdl
begin
  process(clk, sw)
  begin


    if sw(0) = '0' or not ((sw(2)= '0' and sw(3)= '0' and sw(4)= '0')
    or (sw(2)= '0' and sw(3)= '0' and sw(4)= '1')or (sw(2)= '0' and sw(3)= '1' and sw(4)= '1') ) then
        pulse <= '0';
    elsif clk'event and clk = '1' then
    --99999999
```


```vhdl
      if count >= 49999999 then
        count <= 0;
        pulse <= not pulse;
      elsif sw(1)= '0' or (count=49999998 ) then
        count <= count + 1;
      elsif sw(1)='1' and not(count=49999998) then
        count <= count+2;
      end if;
    end if;
  end process;
  led(4 downto 1)<= sw(4 downto 1);

  led(0) <= pulse;
end Behavioral;
```

After completing code I run synthesis and get project elaborated design. This is the project schematic:



Utilization and usage of devices are:

Resource	Estimation	Available	Utilization %
LUT	17	20800	0.08
FF	27	41600	0.06
IO	11	106	10.38
BUFG	1	32	3.13

After getting synthesis design, I get implementation utilization and usage devices table:

Resource	Utilization	Available	Utilization %
LUT	16	20800	0.08
FF	27	41600	0.06
IO	11	106	10.38
BUFG	1	32	3.13

Verification

The timing constraints in this project are very loose so we don't need complicated verification. The acceptance test itself is satisfactory.

References

https://github.com/sonebu/elec305-lab1-example/tree/main