香港中文大學
The Chinese University of Hong Kong

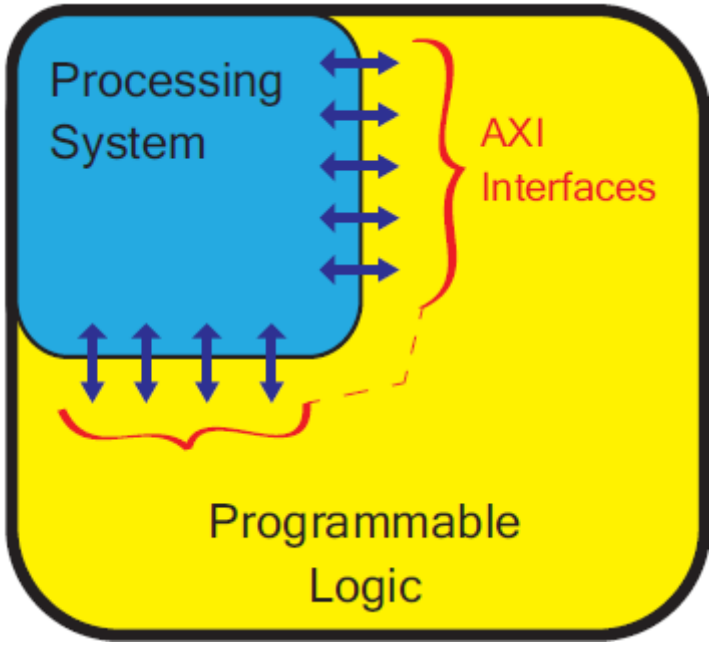
CENG3430 Rapid Prototyping of Digital Systems

Lab04: Stopwatch

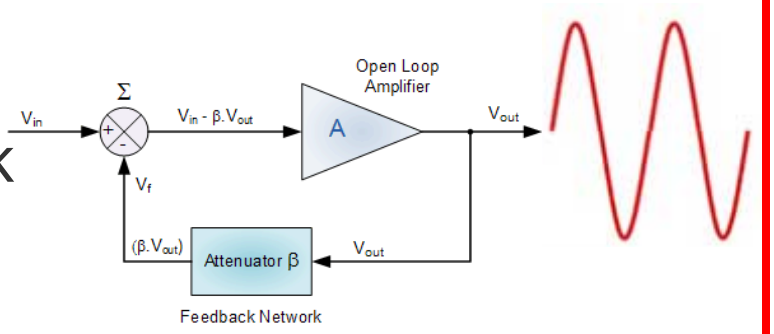
Clock Setup

Clock Sources on ZedBoard

- **Processing System**
  - PS subsystem uses a dedicated 33.3333 MHz clock source with series termination.
    - IC18, Fox 767-33.333333-12
  - PS subsystem can generate up to four phase-locked loop (PLL) based clocks for the PL system.



- **Programmable Logic**
  - An on-board 100 MHz oscillator supplies the PL subsystem clock input on bank 13, pin Y9.
    - IC17, Fox 767-100-136



http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf
https://www.electronics-tutorials.ws/oscillator/oscillators.html

Clock Sources on ZedBoard

- To use the on-board 100 MHz clock input on bank 13, pin Y9, you need to include the following in your XDC constraint file:

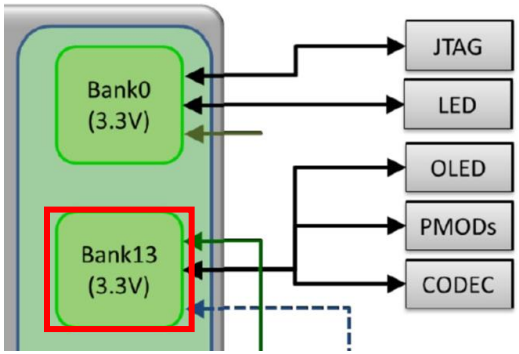
```
set_property IOSTANDARD LVCMOS33 [get_ports clk];
set_property PACKAGE_PIN Y9 [get_ports clk];
create_clock -period 10 [get_ports clk];
```

You can copy this

Note:

*The constraint **`-period 10`** is only used to inform the tool that clock period is 10 ns (i.e., 100 MHz).*

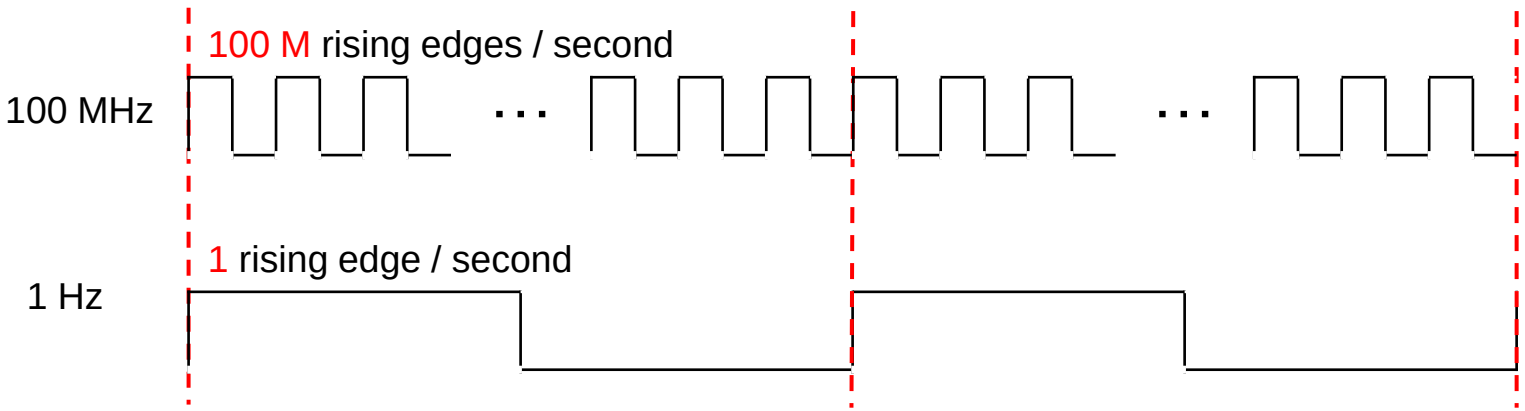
*The constraint **`-period 10`** is **NOT** used specify or generate a different clock period from a given clock source.*



http://zedboard.org/content/changing-frequency-clock-using-createclock

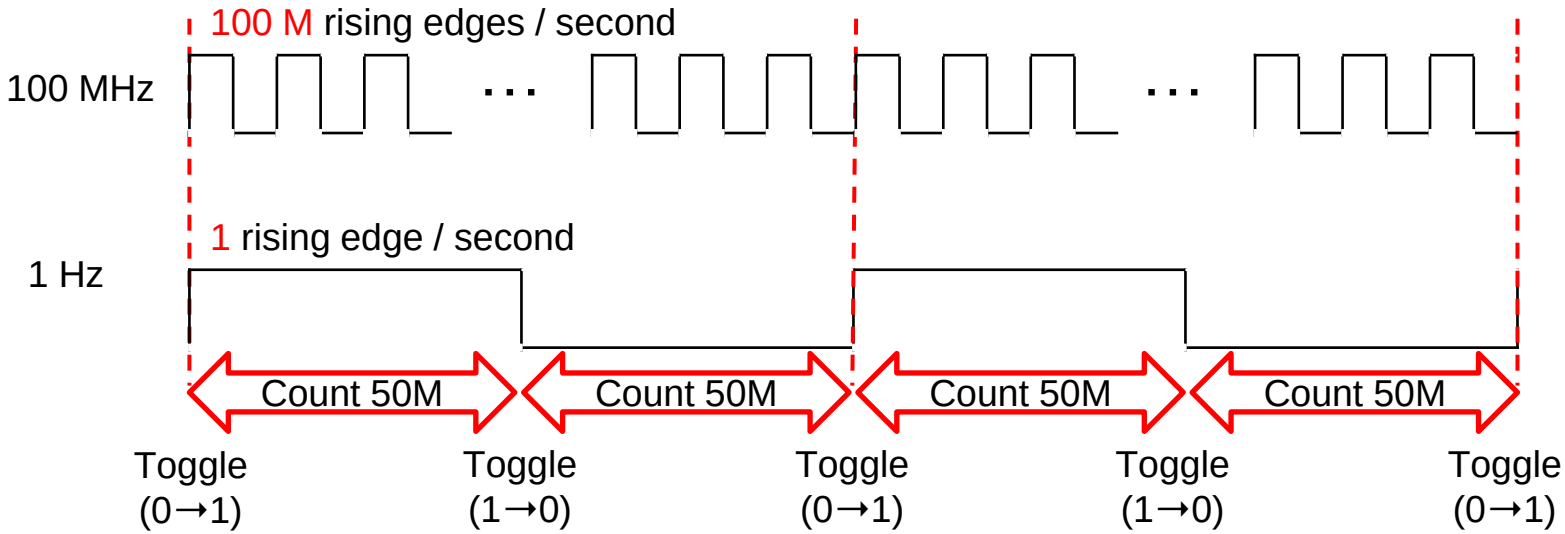
Clock Divider Concept

- However, the 100MHz clock does not always match our needs (e.g., 1Hz)
- **Thus, we need to implement a clock divider**
  - Concept of creating a clock divider (e.g., 100MHz to 1Hz)
    - The clock divider generates 1 rising edge for every 100 M counts.
    - It also generates a falling edge right before the next rising edge.
      - Typically, the falling edge is positioned in the middle (50 M counts here).



Clock Divider Implementation

- Implement a clock divider (100MHz to 1Hz)
  - Toggle the output signal in every 50M counts



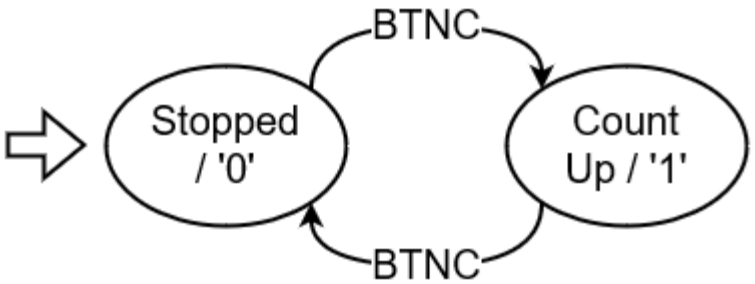
Core codes

```
if (rising_edge(CLK_IN)) then
            if(counter = 50000000 - 1) then
                sig <= NOT sig;
                counter <= 0;
            else
                counter <= counter + 1;
            end if;
end if;
```

Example

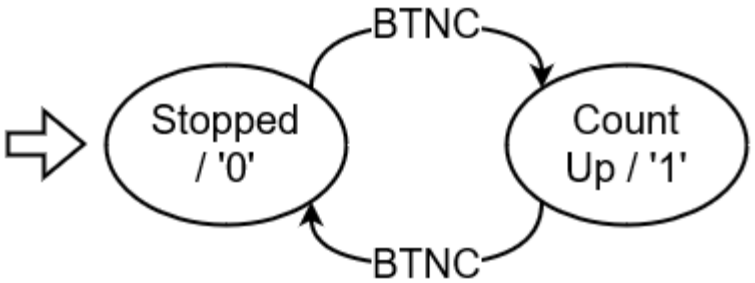
Up Count Stopwatch

- **Behavior**
  - **Two states**
    - **Stopped State** (Initial State)
    - **Count-Up State**
  - Pressing BTNC for state transition
    - BTNC should follow the 4Hz clock rising edge
  - LED0 ~ 3 display the value in both states
  - LED7 display the state value



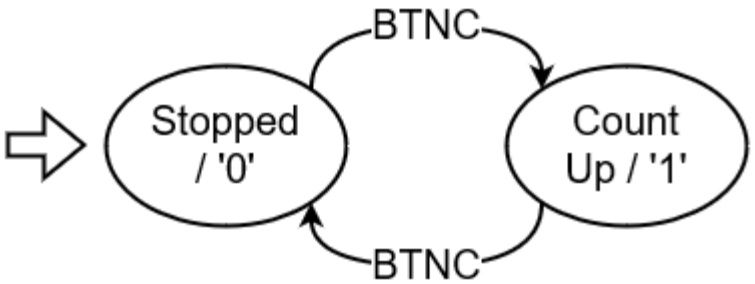
Stopwatch Behavior

- **Behavior**
  - **Two states**
    - **Stopped State** (Initial State)
      - Select the value using **SW0 ~ 3**
      - Display the value in binary format through **LED0 ~ 3**
    - **Count-Up State**



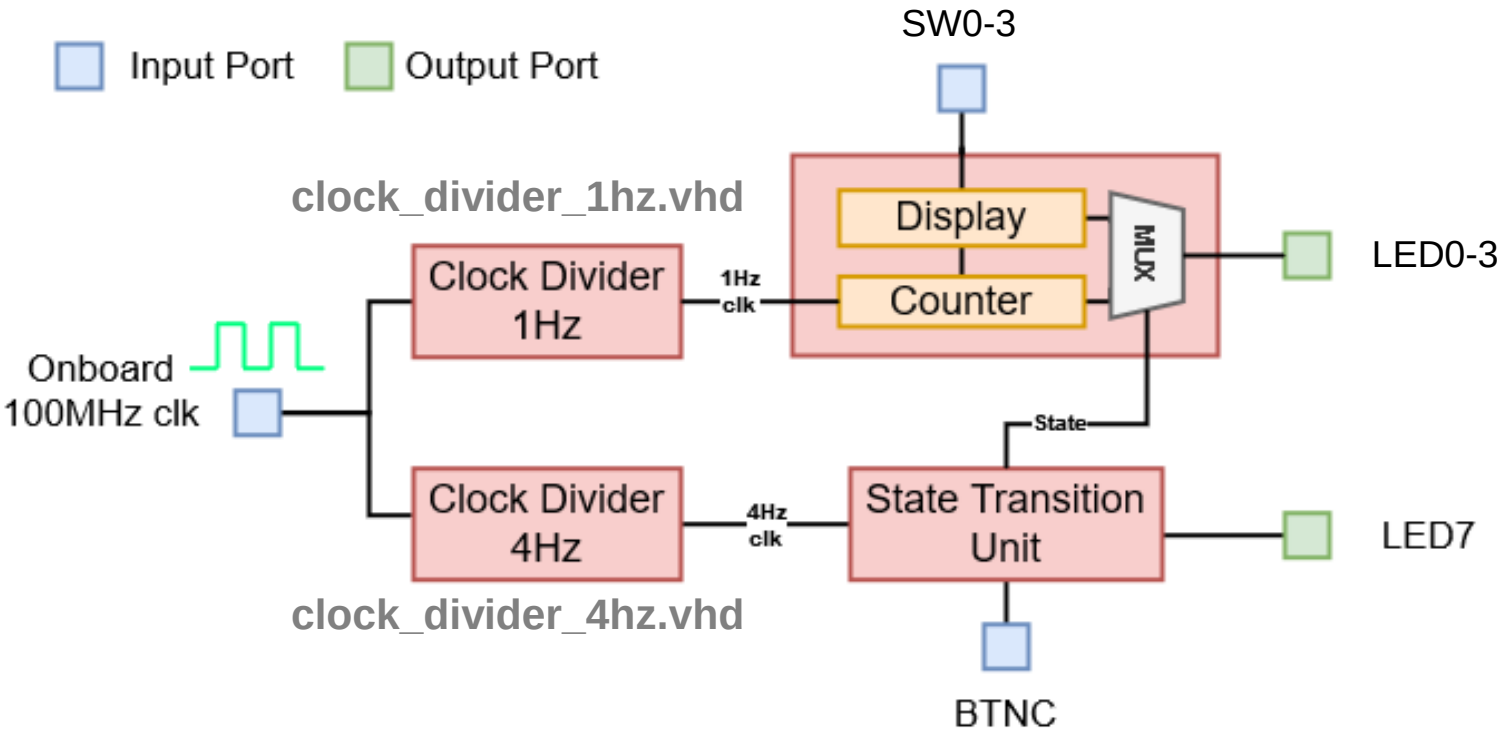
Stopwatch Behavior

- **Behavior**
  - **Two states**
    - **Stopped State**
    - **Count-Up State**
      - Count up from the value selected in Stopped State in 1Hz
      - If the value reaches “1111”, jump to “0000” and keep count up



Stopwatch Behavior

- **Logic Schematic**



lab04.vhd

Step by Step Implementation

lab04.vhd

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

entity lab04 is
    port(
        CLK : IN STD_LOGIC;
        DIN : IN STD_LOGIC_VECTOR(3 downto 0);
        BTNC : IN STD_LOGIC;
        STATE : OUT STD_LOGIC;
        OUTPUT : OUT STD_LOGIC_VECTOR(3 downto 0)
    );
end lab04;

architecture Behavioral of lab04 is
begin

end Behavioral;
```

Fill in the input and output port according to the logic schematic

Step by Step Implementation

lab04.vhd

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

entity lab04 is
    ...

architecture Behavioral of lab04 is
    component clock_divider_1hz is
    port(
        CLK_IN: IN STD_LOGIC;
        CLK_OUT: OUT STD_LOGIC
    );
    end component;

    component clock_divider_4hz is
    port(
        CLK_IN: IN STD_LOGIC;
        CLK_OUT: OUT STD_LOGIC
    );
    end component;
begin
end Behavioral;
```

As the onboard clock source is 100MHz, create two clock divider components

One is for the 1Hz clock, the second is for the 4Hz clock

(See the next page for implementation)

Step by Step Implementation

Create two .vhd file to the design sources
(For 1Hz and 4Hz clock dividers)

clock_divider_1hz.vhd

```
entity clock_divider_1hz
  is  Port (
    CLK_IN : IN STD_LOGIC;
    CLK_OUT : OUT STD_LOGIC
  );
end clock_divider_1hz;

architecture Behavioral of clock_divider_1hz is
    signal counter : integer := 0;
    signal sig : STD_LOGIC := '0';
begin

    CLK_OUT <= sig;

    process(CLK_IN)
    begin
        if (rising_edge(CLK_IN)) then
            if(counter = 50000000 - 1) then
                sig <= NOT sig;
                counter <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;

end Behavioral;
```

100MHz clk => 100,000,000 clock cycles in 1 second

For 1Hz clk, toggle when count = 50,000,000
(50,000,000 low – 50,000,000 high)

clock_divider_4hz.vhd

```
entity clock_divider_4hz
  is  Port (
    CLK_IN : IN STD_LOGIC;
    CLK_OUT : OUT STD_LOGIC
  );
end clock_divider_4hz;

architecture Behavioral of clock_divider_4hz is
    signal counter : integer := 0;
    signal sig : STD_LOGIC := '0';
begin

    CLK_OUT <= sig;

    process(CLK_IN)
    begin
        if (rising_edge(CLK_IN)) then
            if(counter = 12500000 - 1) then
                sig <= NOT sig;
                counter <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;

end Behavioral;
```

Similarly, 100MHz clk => 25,000,000 clock cycles in 0.25 seconds

For 4Hz clk, toggle when count = 12,500,000
(12,500,000 low – 12,500,000 high)

See Page 5 for the concept of the clock divider

Step by Step Implementation

lab04.vhd

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

entity lab04 is
    ...

architecture Behavioral of lab04 is
    ...
    signal clk_1hz : STD_LOGIC;
    signal clk_4hz : STD_LOGIC;
begin
    CD_1Hz: clock_divider_1hz
        port map(CLK, clk_1hz);

    CD_4Hz: clock_divider_4hz
        port map(CLK, clk_4hz);
end Behavioral;
```

Go back to the lab04.vhd

Port map the clock divider twice

Create the following signals as output

1. clk_1hz
2. clk_4hz

Step by Step Implementation

lab04.vhd

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

entity lab04 is
    ...

architecture Behavioral of lab04 is
    ...
    signal sig_state: std_logic := '0';
begin
    ...
    process(clk_4Hz)
    begin
        if rising_edge(clk_4Hz) then
            if BTNC = '1' then
                sig_state <= NOT sig_state;
            else
                sig_state <= sig_state;
            end if;
        end if;
    end process;
end Behavioral;
```

Create a process for state transition

By synchronizing the BTNC with the 4Hz clock

- The button signal is more predictable for the circuit
- Reduce the possibility of glitching due the high sensibility

Step by Step Implementation

lab04.vhd

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

entity lab04 is
    ...

architecture Behavioral of lab04 is
    ...
    signal counter : STD_LOGIC_VECTOR(3 downto 0);
begin
    ...
    process(sig_state, clk_1hz)
    begin
        if sig_state = '0' then
            counter <= DIN;
        else
            ...
        end if;
    end process;
end Behavioral;
```

For state 0, simply set the counter to the switches input

For state 1, see next page

Step by Step Implementation

lab04.vhd

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

entity lab04 is
    ...

architecture Behavioral of lab04 is
    ...
begin
    ...
    process(sig_state, clk_1hz)
    begin
        if sig_state = '0' then
            counter <= DIN;
        else
            if rising_edge(clk_1hz) then
                counter <= std_logic_vector(unsigned(counter)+1);
            end if;
        end if;
    end process;
end Behavioral;
```

Add this library for integer arithmetic and type conversion

For state 1, count up the counter by 1Hz clock

The counter (std_logic_vector)

1. Convert into unsigned integer
2. Add 1
3. Convert back to std_logic_vector

(If counter reaches “1111”, it will jump to 0 when add 1 since it is unsigned integer)

Step by Step Implementation

lab04.vhd

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

entity lab04 is
    ...

architecture Behavioral of lab04 is
    ...
begin
    ...
    OUTPUT <= counter;
    STATE <= sig_state;
end Behavioral;
```

Use concurrent statement to connect the outputs with the corresponding signals

Implementation (Complete Code)

lab04.vhd

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

entity lab04 is
  Port (
    CLK : IN STD_LOGIC;
    DIN : IN STD_LOGIC_VECTOR(3 downto 0);
    BTNC : IN STD_LOGIC;
    STATE : OUT STD_LOGIC;
    OUTPUT : OUT STD_LOGIC_VECTOR(3 downto 0)
  );
end lab04;

architecture Behavioral of lab04 is
    component clock_divider_1hz is
      Port (
        CLK_IN : IN STD_LOGIC;
        CLK_OUT : OUT STD_LOGIC
      );
    end component;
    component clock_divider_4hz is
      Port (
        CLK_IN : IN STD_LOGIC;
        CLK_OUT : OUT STD_LOGIC
      );
    end component;

    signal clk_1hz : STD_LOGIC;
    signal clk_4hz : STD_LOGIC;
    signal sig_state : STD_LOGIC := '0';
    signal counter : STD_LOGIC_VECTOR(3 downto
0);

begin

    CD_1Hz: clock_divider_1hz
        port map(CLK, clk_1hz);

    CD_4Hz: clock_divider_4hz
        port map(CLK, clk_4hz);

    process(clk_4hz)
    begin
        if rising_edge(clk_4hz) then
            if BTNC = '1' then
                sig_state <= NOT sig_state;
            else
                sig_state <= sig_state;
            end if;
        end if;
    end process;
```

Clock Dividers

State Transition Unit

Continue…

Implementation (Complete Code)

```
process(sig_state, clk_1hz)
    begin
        if sig_state = '0' then
            counter <= DIN;
        else
            if rising_edge(clk_1hz) then
                counter <= std_logic_vector(unsigned(counter)+1);
            end if;
        end if;
end process;

    OUTPUT <= counter;
    STATE <= sig_state;

end Behavioral;
```

State Output (Count up)

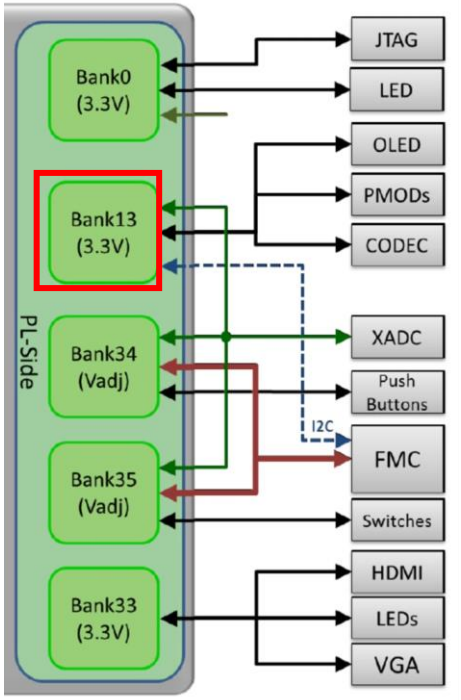
Output Connection

You can copy this

- Highly recommend you implement the two-state stopwatch step by step from Page 7 to understand the logic schematic in Page 6
- For state machine, a good convention is to separate the state transition and the state behavior into different process
  - (For more complex design, the state behavior will cut into multiple processes)

Constraints File (XDC)

- LED7 : STATE
- BTNC : BTNC
- SW0~3 : DIN
- LED0~3 : OUTPUT
- On board clock source : CLK
  - On board 100MHz clock is pin Y9
    - Page 4 provides the complete xdc code



2.5 Clock sources

The Zynq-7000 AP SoC's PS subsystem uses a dedicated 33.3333 MHz clock source, IC18, Fox 767-33.333333-12, with series termination. The PS infrastructure can generate up to four PLL-based clocks for the PL system. An on-board 100 MHz oscillator, IC17, Fox 767-100-136, supplies the PL subsystem clock input on bank 13, pin Y9.

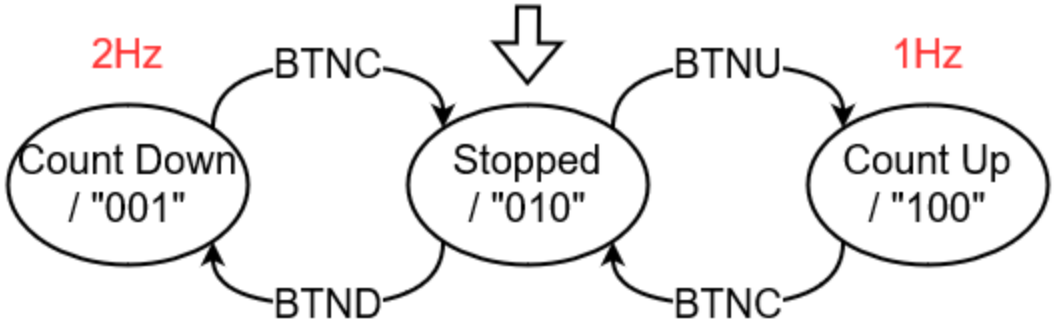
p.18, ZedBoard_HW_UG_v2_2

A short demo video of the two-state stopwatch is uploaded to the blackboard, feel free to check the behavior

Task

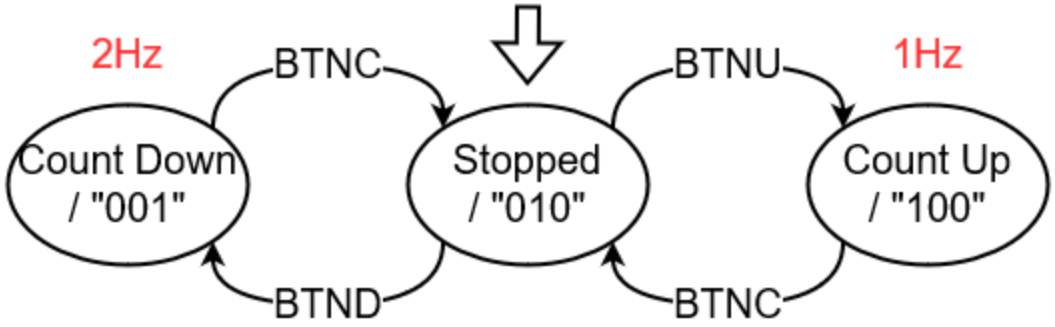
Three-state Stopwatch

- Compare to the two-state stopwatch in example
  - **Stopped state (“010”)** is the same
  - **Count Up state (“100”)** is no longer loop back
    - If the counter reaches “1111”, stops at “1111”
  - **Count Down state (“001”)**
    - Count down the value selected by the stopped state in 2Hz
    - If the counter reaches “0000”, stops at “0000”



Three-state Stopwatch

- Compare to the two-state stopwatch in example
  - **LED 5-7** for showing the new state values (“001”, “010”, “100”)
  - **BTNU and BTND** are added for state transition
    - BTNC will no longer switch back from Count Up state to Stopped state
    - Non-specified transition will create no effects
      - E.g., Pressing BTND in Count Up state has no effect

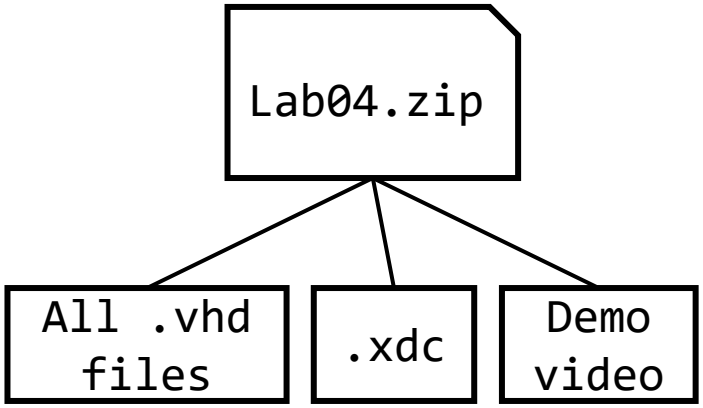


Design Flow

- Recommended Design Flow
  (Assuming the example is already implemented correctly)
  1. Modify the Count Up State to stop at “1111”
     - Test the step 1 on-board and debug if needed
  2. Create the 2Hz clock
  3. Complete the state transition process of three states
     - Unlike the example, useable buttons are different in each state (The entire process will be different)
     - Test the LED5-7 of state transition with buttons
  4. Complete the state behavior process with all three states
     - For robustness, please implement all the states and else case

Submission Guide

- Following the specification starting from Page 24 to implement the Three-State Stopwatch
- Demo video instruction
  1. Select 1010b and press BTNU
  2. Wait until 1111b and press BTND
  3. Then press BTNC
  4. Select 1000b and press BTND
  5. Wait until 0000b and press BTNU
  6. Finally press BTNC



- Submit the zip file according to the figure
  - The video should clearly demonstrate LEDs, SWs and BTNs at the same time (A demo video is provided in the blackboard)
  - Deadline: **12:30 on 19 Feb. 2025**
    - Late submission is NOT acceptable.

香港中文大學
The Chinese University of Hong Kong

Thank You!