

นายภฤศ ตัณฑ์วรกุล 6601012610083

นายวิวัฒน์ ธิติเชษฐตระกุล 6601012610148

รายงานโครงงาน

เรื่อง นาฬิกาดิจิตอล

นาฬิกาดิจิตอลเป็นนาฬิกาชนิดหนึ่งที่ใช้วงจรดิจิตอลในการสร้างขึ้นมา โดยในรายงานนี้จะนำเสนอขั้นตอนวิธีการทำนาฬิกาดิจิตอลด้วย FPGA DE10-Lite (Altera MAX10 10M50DAF484C7G) และโปรแกรมด้วยภาษา VHDL (VHSIC Hardware Description Language) ผ่าน Quartus Prime 20.1 Lite edition

Youtube: <https://www.youtube.com/watch?v=gMObAGrbG9w>

Git hub: https://github.com/Nextjingjing/digital_clock_fpga

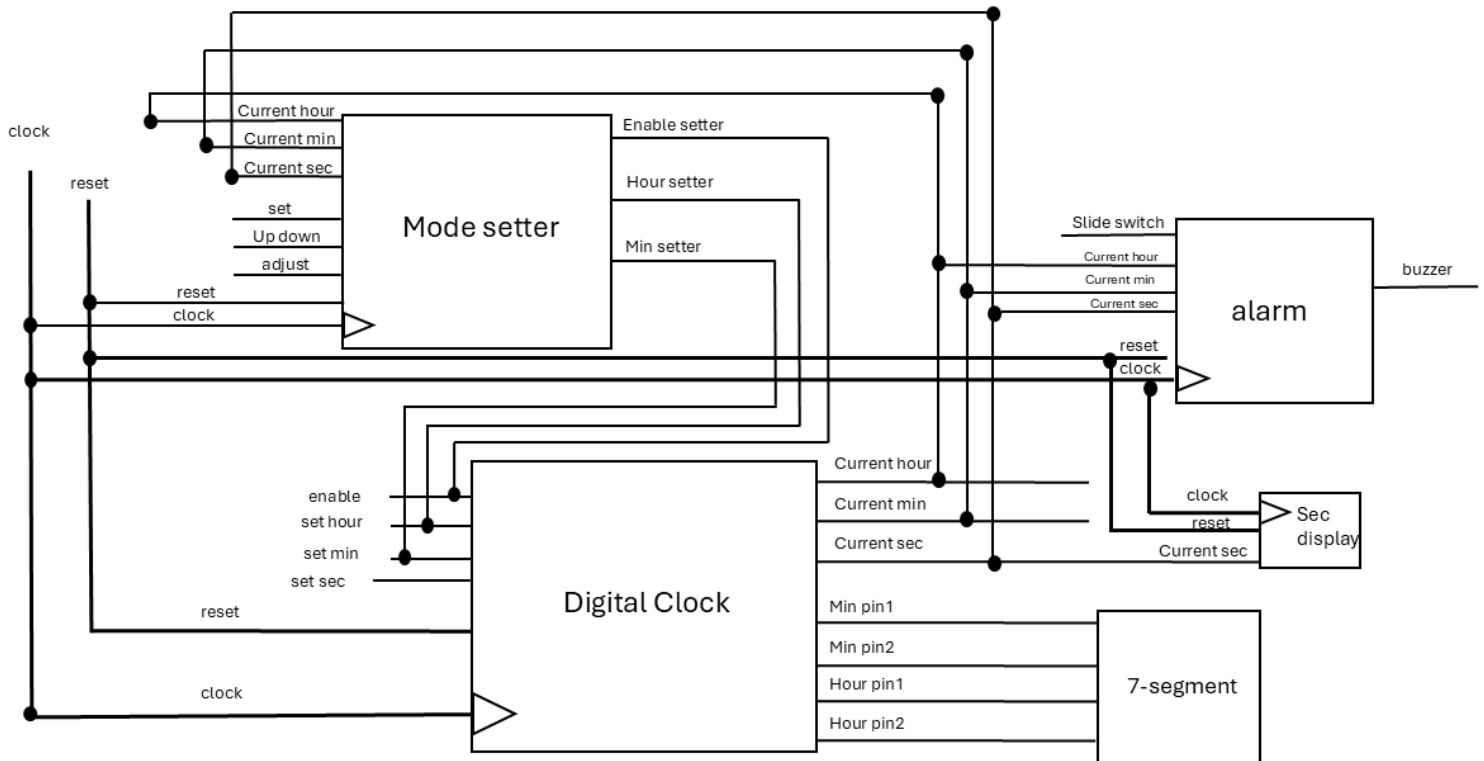
โดยตัวนาฬิกาของรายงานนี้มีฟังก์ชันการทำงานต่อไปนี้



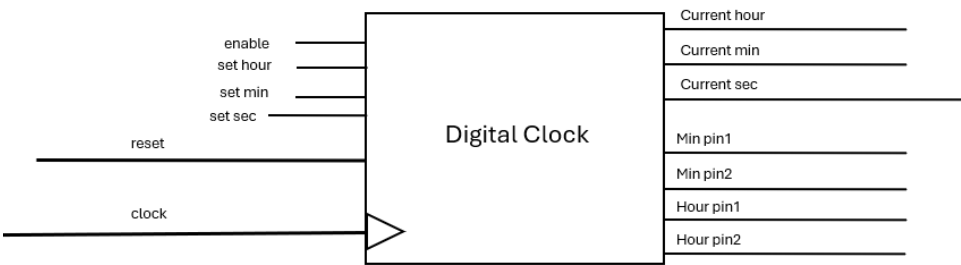
- แสดงผลเวลาทาง 7-segment เป็นชั่วโมง, นาที และวินาที
- การปรับเวลาในหน่วยชั่วโมง และนาที
- การแจ้งเตือนทุก 15 นาที หรือทุก 1 ชั่วโมง โดยสามารถปรับได้
- การรีเซ็ตการทำงานของวงจร

แผนภาพ Block Diagram และการออกแบบ

แผนภาพนี้แสดงถึงการออกแบบวงจรดิจิทัล โดยเราอาศัยหลักการของวงจรซีควเอนเชียล (Sequential Logic) หรือเครื่องสถานะจำกัด (Finite State Machine: FSM) และทำงานแบบวงจรซิงโครนัส (Synchronous) ใช้สัญญาณ clock ในการกำหนดจังหวะการทำงานของหน่วยความจำทั้งหมด เพื่ออัปเดตสถานะพร้อมกัน



วงจร Digital Clock



วงจรนี้มีหน้าที่เก็บค่าสถานะของเวลาปัจจุบัน และคอยอัปเดตเวลาให้เปลี่ยนแปลงตลอดเวลา โดยอาศัยหลักการของวงจรรนับ (Counter Circuit) โดยหากทราบถึงความถี่ของสัญญาณนาฬิกา จะทราบได้ว่า จะต้องนับไปจนถึงเท่าไรจึงจะครบเวลา 1 วินาทีโดยอาศัยสมการนี้

$1\text{ sec} = \text{clock frequency(Hz)} - 1$

ดังนั้นหากต้องการ 1 วินาทีที่จะต้องนับไปจนถึง **clock frequency - 1** จึงจะครบเวลา ต่อไปจะต้องมีรีจิสเตอร์ในการจดจำสถานะดังนี้ ชั่วโมง นาที และวินาที และคอยอัปเดตรีจิสเตอร์ตามหลักหน่วยเวลาคือ 1 ชั่วโมงมี 60 นาทีและ 1 นาทีมี 60 วินาที และในวงจรนี้เราจะมีคำนวณ BCD ของแต่ละหลักของนาที และชั่วโมงเพื่อนำไปนำเสนอที่ 7-segment ต่อไป

อินพุต และเอาต์พุต


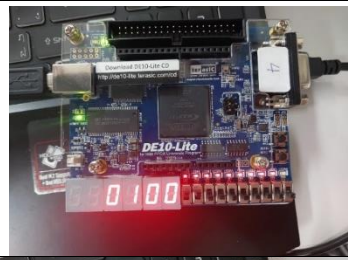

อินพุตของวงจรนี้

ชื่อของอินพุต	คำอธิบาย
enable	ให้อนุญาตให้เกิดการปรับเปลี่ยนค่าของเวลาได้
Set hour	ปรับเปลี่ยนเวลาหน่วยชั่วโมงตามตัวเลข
Set min	ปรับเปลี่ยนเวลาหน่วยนาทีตามตัวเลข
Set sec	ปรับเปลี่ยนเวลาหน่วยวินาทีตามตัวเลข
reset	ล้างค่าสถานะทุกอย่าง
clock	สัญญาณนาฬิกาเพื่อให้เกิดการทำงานที่พร้อมกัน

เอาต์พุตของวงจรนี้

ชื่อของเอาต์พุต	คำอธิบาย
Current hour	ค่าสถานะเวลาชั่วโมง
Current min	ค่าสถานะเวลานาที
Current sec	ค่าสถานะเวลาวินาที
Min pin1	BCD หลักหน่วยของนาฬิกา
Min pin2	BCD หลักสิบของนาฬิกา
Hour pin1	BCD หลักหน่วยของชั่วโมง
Hour pin2	BCD หลักสิบของชั่วโมง

การทดสอบ

ภาพการทดลอง	คำอธิบาย
	การทดสอบการอัปเดตนาฬิกา ขั้นตอนการทำการทดลองให้เปลี่ยนค่าพารามิเตอร์ให้ 1 นาฬิกาที่มีแค่ 5 วินาที เพื่อความสะดวกในการทดลอง
	การทดสอบการอัปเดตชั่วโมง ขั้นตอนการทำการทดลองให้เปลี่ยนค่าพารามิเตอร์ให้ 1 ชั่วโมงมีแค่ 5 นาที และแต่ละนาฬิกาที่มี 5 วินาที เพื่อความสะดวกในการทดลอง
	การทดสอบการกดรีเซ็ตของวงจร

โค้ด VHDL

```

library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_std.all;

entity Digital_clock is

    generic (

        ClockFreq : integer := 10000000 -- 10 MHz clock frequency
    );

    port (

        Clock      : in std_logic;

        reset      : in std_logic;

        Seconds    : out integer;

        Minutes    : out integer;

        Hours      : out integer;

        Set_Hours  : in integer := 0;

        Set_Minutes : in integer := 0;

        Set_Seconds : in integer := 0;

        Set_Enable  : in std_logic := '0';

        Min_pin1   : out std_logic_vector(3 downto 0) := "0000";

        Min_pin2   : out std_logic_vector(3 downto 0) := "0000";

        Hur_pin1   : out std_logic_vector(3 downto 0) := "0000";

        Hur_pin2   : out std_logic_vector(3 downto 0) := "0000"
    );

end entity Digital_clock;

```

architecture behavior of Digital_clock is

```

    signal counts      : integer := 0;

    signal internal_seconds : integer := 0;

    signal internal_minutes : integer := 0;

    signal internal_hours  : integer := 0;

```

begin

-- Assign internal signals to output ports

Seconds <= internal_seconds;

Minutes <= internal_minutes;

Hours <= internal_hours;

process(Clock)

begin

if rising_edge(Clock) then

if reset = '0' then

-- Reset all internal values to 0

counts <= 0;

internal_seconds <= 0;

internal_minutes <= 0;

internal_hours <= 0;

Min_pin1 <= "0000";

Min_pin2 <= "0000";

Hur_pin1 <= "0000";

Hur_pin2 <= "0000";

elsif Set_Enable = '1' then

-- Setting Mode: Update internal time based on Set inputs

internal_hours <= Set_Hours mod 24;

internal_minutes <= Set_Minutes mod 60;

internal_seconds <= Set_Seconds mod 60;

```

-- Update display pins based on set values

Min_pin1 <= std_logic_vector(to_unsigned(Set_Minutes mod 10,
4));

Min_pin2 <= std_logic_vector(to_unsigned(Set_Minutes / 10, 4));
Hur_pin1 <= std_logic_vector(to_unsigned(Set_Hours mod 10, 4));
Hur_pin2 <= std_logic_vector(to_unsigned(Set_Hours / 10, 4));

else

-- Normal Counting Mode

if counts = ClockFreq - 1 then

counts <= 0;


if internal_seconds = 59 then

internal_seconds <= 0;


if internal_minutes = 59 then

internal_minutes <= 0;


Min_pin1 <= "0000";

Min_pin2 <= "0000";


if internal_hours = 23 then

internal_hours <= 0;


Hur_pin1 <= "0000";

Hur_pin2 <= "0000";

else

internal_hours <= internal_hours + 1;


-- Update display pins for hours

Hur_pin1 <= std_logic_vector(to_unsigned((internal_hours
+ 1) mod 10, 4));

Hur_pin2 <= std_logic_vector(to_unsigned((internal_hours
+ 1) / 10, 4));

end if;

```

```

else

internal_minutes <= internal_minutes + 1;


-- Update display pins for minutes

Min_pin1 <= std_logic_vector(to_unsigned((internal_minutes
+ 1) mod 10, 4));

Min_pin2 <= std_logic_vector(to_unsigned((internal_minutes
+ 1) / 10, 4));

end if;

else

internal_seconds <= internal_seconds + 1;

end if;

else

counts <= counts + 1;

end if;

end if;

end process;

end architecture behavior;

```

วงจร Sec display



วงจรนี้จะทำการแสดงไฟกระพริบ 0.1 วินาที เมื่อมีผ่านไปทุกๆ 1 วินาที โดยใช้การดูว่าวินาทีปัจจุบันตรงกับวินาทีในรีจิสเตอร์หรือไม่ ถ้าหากว่าตรงกันแสดงว่าวินาทีไม่เพิ่ม แต่ถ้าไม่ตรงกันแสดงว่าเวลาผ่านไป 1 วินาทีแล้วและจะสร้างสัญญาณ **pulse** ขึ้น 0.1 วินาทีโดยวงจรนับ (Counter Circuit) โดยสมการ

$$1 \text{ sec} = \frac{\text{clock frequency(Hz)}}{10} - 1$$

โดยวงจรจะตรวจสอบทุกๆ ขอบขาขึ้นของนาฬิกา

อินพุต และเอาต์พุต

อินพุตของวงจรนี้

ชื่อของอินพุต	คำอธิบาย
clock	สัญญาณนาฬิกาเพื่อให้เกิดการทำงานที่พร้อมกัน
reset	ล้างค่าสถานะการทำงานทุกอย่าง
Current sec	เวลาปัจจุบันเพื่อการเปรียบเทียบเวลาว่ามีการเปลี่ยนแปลง

เอาต์พุตของวงจรนี้

ชื่อของเอาต์พุต	คำอธิบาย
Led out	Pulse สัญญาณ 0.1 วินาที

```

library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_std.all;

entity Led_Control is
    generic(
        ClockFreq : integer := 10000000 -- Clock frequency
    );
    port(
        Clock   : in std_logic;
        reset   : in std_logic;
        seconds  : in integer;    -- Input seconds
        led_out  : out std_logic  -- LED output (active low)
    );
end entity Led_Control;

architecture Behavioral of Led_Control is
    -- Calculate PulseTime in the architecture body

    constant PulseTime : integer := ClockFreq / 10; -- 0.1 second pulse
    duration

    signal prev_seconds : integer := 0; -- Stores previous value of
seconds

    signal led_timer   : integer := 0; -- Timer for 0.1 second pulse

    signal led_active  : std_logic := '0'; -- Internal signal to control LED

begin
    process(Clock)
    begin
        if rising_edge(Clock) then

            if reset = '0' then

                prev_seconds <= 0;

                led_timer   <= 0;

                led_active  <= '0';

                led_out     <= '1'; -- LED off during reset (active low)

```

```

            else

                -- Detect change in seconds

                if seconds /= prev_seconds then

                    prev_seconds <= seconds;

                    led_active <= '1'; -- Activate LED when seconds change

                    led_timer  <= 0; -- Reset timer

                end if;

                -- Control LED timing (0.1 second pulse)

                if led_active = '1' then

                    if led_timer < PulseTime then

                        led_timer <= led_timer + 1;

                        led_out  <= '0'; -- Turn LED on (active low)

                    else

                        led_active <= '0'; -- Deactivate LED after 0.1 seconds

                        led_out  <= '1'; -- Turn LED off (active low)

                    end if;

                else

                    led_out <= '1'; -- Ensure LED is off when not active
(active low)

                end if;

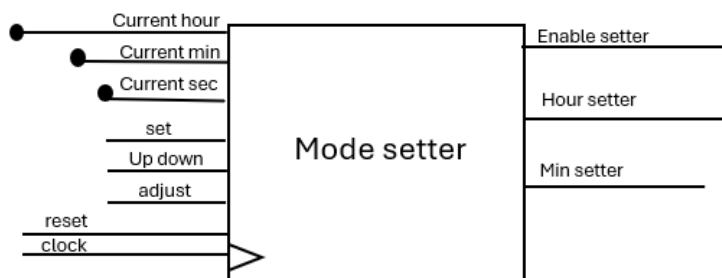
            end if;

        end if;

    end process;
end architecture Behavioral;

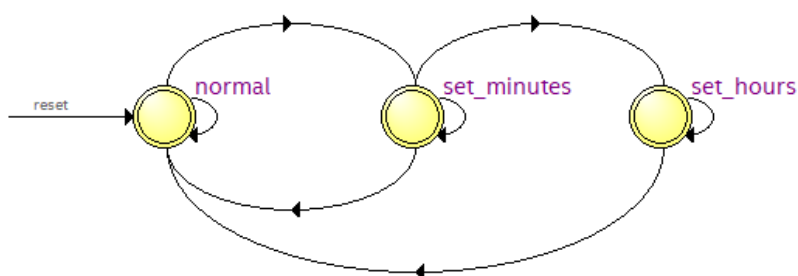
```


วงจร Mode setter



วงจรนี้จะทำหน้าที่ในการเปลี่ยนโหมดของนาฬิกาสำหรับแต่ละโหมดจะเป็นการตั้งค่าเวลาโดยที่จะแบ่งออกเป็น 3 โหมดดังนี้ เมื่อไม่ได้กดปุ่มนาฬิกาจะทำงานตามปกติเมื่อมีการกดปุ่ม 1 ครั้ง วงจรจะเข้าสู่โหมดการตั้งเวลาในหน่วยนาที่และเมื่อกดปุ่มอีก 1 ครั้งวงจรจะเข้าสู่โหมดการตั้งค่าในหน่วยชั่วโมง และเมื่อมีการกดปุ่มอีก 1 ครั้งวงจรจะกลับเข้าสู่การทำงานในโหมดปกติ

Set =1 or reset =0



State diagram ของวงจร

อินพุต และเอาต์พุต

อินพุตของวงจรนี้

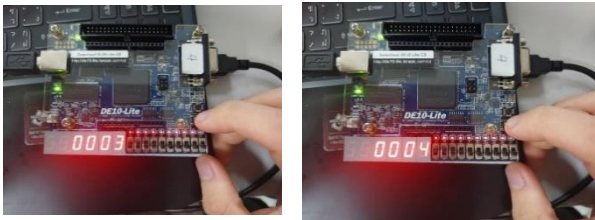
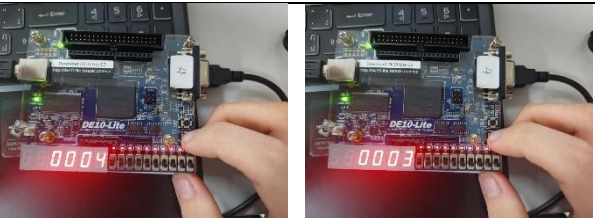
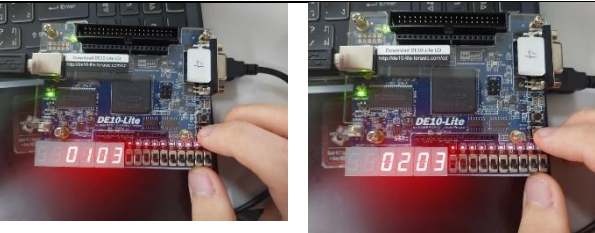
ชื่อของอินพุต	คำอธิบาย
Current hour	ค่าสถานะเวลาชั่วโมง
Current min	ค่าสถานะเวลานาที
Current sec	ค่าสถานะเวลาวินาที
set	เปลี่ยนโหมดของวงจร
Up down	รับค่ามาจาก switch ถ้า 1 จะเพิ่มเวลา 0 จะลดเวลา




adjust	ปรับค่าเพิ่ม/ลงทีละ 1
reset	ล้างค่าสถานะการทำงานทุกอย่าง
clock	สัญญาณนาฬิกาเพื่อให้เกิดการทำงานที่พร้อมกัน

เอาต์พุตของวงจรนี้

ชื่อของเอาต์พุต	คำอธิบาย
Enable setter	ถ้าสถานะเป็น 1 จะปรับเวลาได้
Hour setter	ปรับเวลาเป็นชั่วโมง
Min setter	ปรับเวลาเป็นนาที

การทดสอบ

ภาพการทดลอง	คำอธิบาย
	การทดสอบการตั้งเวลาในหน่วยนาฬิกา ขั้นตอนการทดลอง กดปุ่ม 1 ครั้งเพื่อเข้าสู่โหมดการตั้งค่าในหน่วยนาฬิกาและมีการเลื่อน Slide switch ขึ้นและมีการกดปุ่มจะพบว่าเวลาจะค่อยๆปรับขึ้น
	การทดสอบการตั้งเวลาในหน่วยนาฬิกา ขั้นตอนการทดลอง กดปุ่ม 1 ครั้งเพื่อเข้าสู่โหมดการตั้งค่าในหน่วยนาฬิกาและมีการเลื่อน Slide switch ลงและมีการกดปุ่มจะพบว่าเวลาจะค่อยๆลดลง
	การทดสอบการตั้งเวลาในหน่วยชั่วโมง ขั้นตอนการทดลอง กดปุ่ม 2 ครั้งเพื่อเข้าสู่โหมดการตั้งค่าในหน่วยชั่วโมงและมีการเลื่อน Slide switch ขึ้นและมีการกดปุ่มจะพบว่าเวลาจะค่อยๆเพิ่มขึ้น

		<p>การทดสอบการตั้งเวลาในหน่วยชั่วโมง ขั้นตอนการทดลอง กดปุ่ม 2 ครั้งเพื่อเข้าสู่โหมดการตั้งค่าในหน่วยชั่วโมงและมีการเลื่อน Slide switch ลงและมีการกดปุ่มจะพบว่าเวลาจะค่อยๆลดลง</p>
		<p>การทดลองการกลับเข้าสู่สถานะปกติขั้นตอนการทดลองคือ เมื่อมีการกดปุ่ม 3 ครั้ง วงจรจะกลับเข้าสู่สถานะการทำงานปกติ</p>

<pre> library ieee; use ieee.std_logic_1164.all; use ieee.numeric_std.all; entity Mode_setter is port (Clock : in std_logic; reset : in std_logic; Set : in std_logic; Adjust : in std_logic; Up_Down : in std_logic; Current_Minutes : in integer; Current_Hours : in integer; setter_minutes : out integer; setter_hours : out integer; Mode : out integer); end entity Mode_setter; </pre>	<p>architecture behavior of Mode_setter is</p> <pre> -- Define the possible states type state_type is (normal, set_minutes, set_hours); signal state : state_type := normal; signal internal_minutes : integer := 0; signal internal_hours : integer := 0; -- Signals for edge detection of Set button signal Set_prev : std_logic := '1'; signal Set_pulse : std_logic := '0'; -- Signals for edge detection of Adjust button signal Adjust_prev : std_logic := '1'; signal Adjust_pulse : std_logic := '0'; -- Debounce signals and counters constant DEBOUNCE_LIMIT : integer := 200000; -- สำหรับ Clock 10 MHz, 200000 เท่ากับ 20 ms signal Set_debounce_cnt : integer := 0; signal Set_debounced : std_logic := '1'; -- เริ่มต้นเป็น '1' (ไม่ถูกกด) signal Adjust_debounce_cnt : integer := 0; signal Adjust_debounced : std_logic := '1'; -- เริ่มต้น เป็น '1' (ไม่ถูกกด) -- Debounce process for Adjust button debounce_adjust: process(Clock) begin if rising_edge(Clock) then if reset = '0' then Adjust_debounce_cnt <= 0; Adjust_debounced <= '1'; else if Adjust = Adjust_debounced then </pre>
---	---

<pre> begin -- Debounce process for Set button debounce_set: process(Clock) begin if rising_edge(Clock) then if reset = '0' then Set_debounce_cnt <= 0; Set_debounced <= '1'; else if Set = Set_debounced then -- ไม่มีการเปลี่ยนแปลง, รีเซ็ตตัวนับ Set_debounce_cnt <= 0; else -- เพิ่มตัวนับ if Set_debounce_cnt < DEBOUNCE_LIMIT then Set_debounce_cnt <= Set_debounce_cnt + 1; else -- เมื่อถึงขีดจำกัดแล้ว ให้เปลี่ยนสถานะ Set_debounced <= Set; Set_debounce_cnt <= 0; end if; end if; end if; end if; end process debounce_set; -- ไม่มีการเปลี่ยนแปลง, รีเซ็ตตัวนับ Adjust_debounce_cnt <= 0; else -- เพิ่มตัวนับ if Adjust_debounce_cnt < DEBOUNCE_LIMIT then Adjust_debounce_cnt <= Adjust_debounce_cnt + 1; else -- เมื่อถึงขีดจำกัดแล้ว ให้เปลี่ยนสถานะ Adjust_debounced <= Adjust; Adjust_debounce_cnt <= 0; end if; end if; end if; end process debounce_adjust; </pre>	<pre> -- Process to detect falling edge of Set button and generate a one-cycle pulse edge_detection_set: process(Clock) begin if rising_edge(Clock) then if reset = '0' then Set_prev <= '1'; Set_pulse <= '0'; else -- Detect falling edge: Set_debounced goes from '1' to '0' if (Set_prev = '1' and Set_debounced = '0') then Set_pulse <= '1'; -- Generate a pulse else Set_pulse <= '0'; end if; -- Update previous Set state Set_prev <= Set_debounced; end if; end if; end process edge_detection_set; -- Process to detect falling edge of Adjust button and generate a one-cycle pulse edge_detection_adjust: process(Clock) begin if rising_edge(Clock) then if reset = '0' then Adjust_prev <= '1'; Adjust_pulse <= '0'; else -- Detect falling edge: Adjust_debounced goes from '1' to '0' if (Adjust_prev = '1' and Adjust_debounced = '0') then Adjust_pulse <= '1'; -- Generate a pulse else Adjust_pulse <= '0'; end if; -- Update previous Adjust state Adjust_prev <= Adjust_debounced; end if; end if; end process edge_detection_adjust; </pre>
---	--

```
-- Main process to handle state transitions and
settings
```

```
state_machine: process(Clock)
```

```
begin
```

```
if rising_edge(Clock) then
```

```
if reset = '0' then
```

```
-- Reset all signals
```

```
state      <= normal;
```

```
internal_minutes <= 0;
```

```
internal_hours  <= 0;
```

```
setter_minutes  <= 0;
```

```
setter_hours    <= 0;
```

```
Mode           <= 0;
```

```
else
```

```
-- Change state only on Set_pulse
```

```
if Set_pulse = '1' then
```

```
case state is
```

```
when normal =>
```

```
state <= set_minutes;
```

```
Mode <= 1;
```

```
when set_minutes =>
```

```
state <= set_hours;
```

```
Mode <= 2;
```

```
when set_hours =>
```

```
state <= normal;
```

```
Mode <= 0;
```

```
when others =>
```

```
null; -- No action
```

```
end case;
```

```
end if;
```

```
-- Handle settings when not in normal state
```

```
if state /= normal then
```

```
if Adjust_pulse = '1' then
```

```
case state is
```

```
when set_minutes =>
```

```
if Up_Down = '1' then
```

```
internal_minutes <=
```

```
(internal_minutes + 1) mod 60;
```

```
else
```

```
if internal_minutes = 0 then
```

```
internal_minutes <= 59;
```

```
else
```

```
internal_minutes <=
```

```
internal_minutes - 1;
```

```
end if;
```

```
end if;
```

```
setter_minutes <= internal_minutes;
```

```
when set_hours =>
```

```
if Up_Down = '1' then
```

```
internal_hours <= (internal_hours +  
1) mod 24;
```

```
else
```

```
if internal_hours = 0 then
```

```
internal_hours <= 23;
```

```
else
```

```
internal_hours <= internal_hours
```

```
- 1;
```

```
end if;
```

```
end if;
```

```
setter_hours <= internal_hours;
```

```
when others =>
```

```
null; -- No action for other states
```

```
end case;
```

```
end if;
```

```
end if;
```

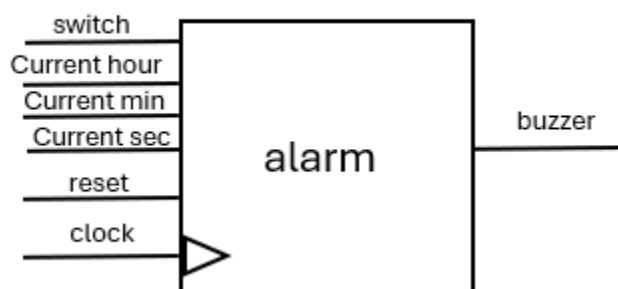
```
end if;
```

```
end if;
```

```
end process state_machine;
```

```
end architecture behavior;
```

วงจร alarm



วงจรนี้มีหน้าที่คือแจ้งเตือนทุกๆ 15 นาที หรือทุกๆ 1 ชั่วโมง โดยจะมีรีจิสเตอร์ที่จดจำค่าเวลาที่ตั้งไว้ และนำไปคำนวณเวลาที่แจ้งเตือน(1 ชั่วโมง หรือ 15 นาที) และจะทำการตรวจทุกๆ ขอบขาขึ้นเมื่อเวลาปัจจุบันตรงกับเวลาที่ได้คำนวณไว้จะแจ้งเตือนเป็นเวลา 3 วินาทีโดยการดูว่า **current sec** เป็น 3 วินาทีแล้วหรือไม่ และจะคำนวณเวลาที่แจ้งเตือนถัดไปเรื่อยๆ และเลือกโหมดโดยใช้ **Switch1, Switch0** โดย (01) คือทุกๆ 15 นาที, (10) คือทุกๆ 1 ชั่วโมง, อื่นๆ จะไม่มีการแจ้งเตือน

อินพุต และเอาต์พุต

อินพุตของวงจรนี้

ชื่อของอินพุต	คำอธิบาย
Switch1, Switch0	เลือกโหมดที่จะแจ้งเตือน
Current hour	ค่าสถานะเวลาหน่วยชั่วโมงปัจจุบัน
Current min	ค่าสถานะเวลาหน่วยนาทีปัจจุบัน
Current sec	ค่าสถานะเวลาหน่วยวินาทีปัจจุบัน
reset	ล้างค่าสถานะทุกอย่าง
clock	สัญญาณนาฬิกาเพื่อให้เกิดการทำงานที่พร้อมกัน

เอาต์พุตของวงจรนี้

ชื่อของเอาต์พุต	คำอธิบาย
Buzzer	แสดงผลการแจ้งเตือน 3 วินาที

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

entity Alarm is

    Port (

        Clock      : in std_logic;

        Reset      : in std_logic; -- Active-low reset signal

        Slide_switch : in std_logic_vector(1 downto 0);

        Hour       : in std_logic_vector(4 downto 0);

        Mins       : in std_logic_vector(5 downto 0);

        Seconds    : in std_logic_vector(5 downto 0); -- New input for
seconds

        Buzzer     : out std_logic

    );

end Alarm;

architecture Behavioral of Alarm is

    signal current_hour   : integer range 0 to 23;

    signal current_min    : integer range 0 to 59;

    signal current_sec    : integer range 0 to 59;

    signal memorized_hour : integer range 0 to 23 := 0;

    signal memorized_min  : integer range 0 to 59 := 0;

    signal alarm_hour     : integer range 0 to 23 := 0;

    signal alarm_min      : integer range 0 to 59 := 0;

    signal Buzzer_sig     : std_logic := '0';

    signal prev_slide_switch : std_logic_vector(1 downto 0) := "00";

    signal alarm_set      : std_logic := '0';

    signal buzzer_on      : std_logic := '0';

```

```

begin

    Buzzer <= Buzzer_sig;

    process(Clock, Reset)

    begin

        if Reset = '0' then -- Active-low reset

            -- Reset all signals and flags

            Buzzer_sig    <= '0';

            alarm_hour    <= 0;

            alarm_min     <= 0;

            memorized_hour <= 0;

            memorized_min <= 0;

            alarm_set     <= '0';

            prev_slide_switch <= "00";

            buzzer_on     <= '0';

        elsif rising_edge(Clock) then

            -- Convert inputs to integers

            current_hour <= to_integer(unsigned(Hour));

            current_min  <= to_integer(unsigned(Mins));

            current_sec  <= to_integer(unsigned(Seconds));

            -- Detect changes in Slide_switch

            if Slide_switch /= prev_slide_switch then

                -- If Slide_switch changed, reset the alarm_set flag

                alarm_set <= '0';

                buzzer_on <= '0';

                Buzzer_sig <= '0';

            end if;

            case Slide_switch is

                when "00" | "11" => -- Memorize current time

                    Buzzer_sig    <= '0';

                    buzzer_on     <= '0';

                    memorized_hour <= current_hour;

                    memorized_min  <= current_min;

                    alarm_set     <= '0'; -- Reset alarm_set when in this state

                when "01" => -- Trigger every 15 minutes

                    if alarm_set = '0' then

                        -- Initialize alarm time based on memorized time

                        alarm_hour <= (memorized_hour + ((memorized_min + 15) / 60)) mod 24;

                        alarm_min  <= (memorized_min + 15) mod 60;

```

```

-- Check if current time matches alarm time

    if (current_hour = alarm_hour) and (current_min = alarm_min) then

        buzzer_on <= '1';

        -- Schedule next alarm

        alarm_hour <= (alarm_hour + ((alarm_min + 15) / 60)) mod 24;

        alarm_min <= (alarm_min + 15) mod 60;

    end if;

when "10" => -- Trigger every 1 hour

    if alarm_set = '0' then

        -- Initialize alarm time based on memorized time

        alarm_hour <= (memorized_hour + 1) mod 24;

        alarm_min <= memorized_min;

        alarm_set <= '1';

    end if;

    -- Check if current time matches alarm time

    if (current_hour = alarm_hour) and (current_min = alarm_min) then

        buzzer_on <= '1';

        -- Schedule next alarm

        alarm_hour <= (alarm_hour + 1) mod 24;

        -- alarm_min remains the same

    end if;

when others =>

    Buzzer_sig <= '0';

    buzzer_on <= '0';

end case;

-- Handle buzzer activation based on Seconds input

if buzzer_on = '1' then

    if current_sec < 3 then

        Buzzer_sig <= '1';

    else

        Buzzer_sig <= '0';

        buzzer_on <= '0'; -- Stop the buzzer after 3 seconds

    end if;

else

    Buzzer_sig <= '0';

end if;

-- Update prev_slide_switch

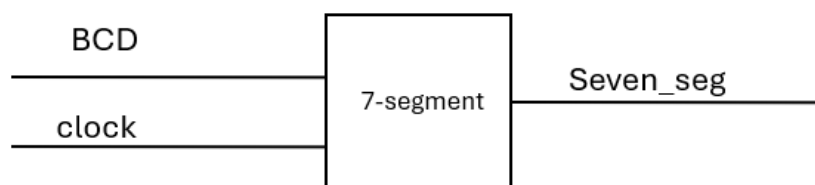
prev_slide_switch <= Slide_switch;

end if;

end process;

```


7-segment



วงจรนี้จะนำค่าผลลัพธ์ที่ได้มาแสดงผลบน 7-segments

อินพุต และเอาต์พุต

อินพุตของวงจรนี้

ชื่อของอินพุต	คำอธิบาย
BCD	รับค่ามาแสดงผลใน 7-Segment
clock	สัญญาณนาฬิกาเพื่อให้เกิดการทำงานที่พร้อมกัน

เอาต์พุตของวงจรนี้

ชื่อของเอาต์พุต	คำอธิบาย
Seven_seg	แสดงผลลัพธ์ของ 7-segments

```

library ieee;
use ieee.std_logic_1164.all;

entity BDC_to_7_segmen is

    port(
        BCD: in std_logic_vector (3 downto 0);
        clk_i : in std_logic;
        seven_seg :out std_logic_vector (6 downto 0));

end BDC_to_7_segmen;

architecture data_process of BDC_to_7_segmen is
begin
    process(clk_i) -- sensitivity list
    begin
        if clk_i'event and clk_i='1' THEN
            case BCD_i is          --gfedcba

                when "0000" => seven_seg <= "1000000"; --7-segment display number 0

            when "0001" => seven_seg <= "1111001"; --7-segment display number 1

            when "0010" => seven_seg <= "0100100"; --7-segment display number 2

            when "0011" => seven_seg <= "0110000"; --7-segment display number 3

            when "0100" => seven_seg <= "0011001"; --7-segment display number 4

            when "0101" => seven_seg <= "0010010"; --7-segment display number 5

            when "0110" => seven_seg <= "0000010"; --7-segment display number 6

            when "0111" => seven_seg <= "1111000"; --7-segment display number 7

            when "1000" => seven_seg <= "0000000"; --7-segment display number 8

            when "1001" => seven_seg <= "0010000"; --7-segment display number 9

            when others => seven_seg <= "0001110"; --7-segment display F

            end case;

        end if;

    end process;
end data_process;

```