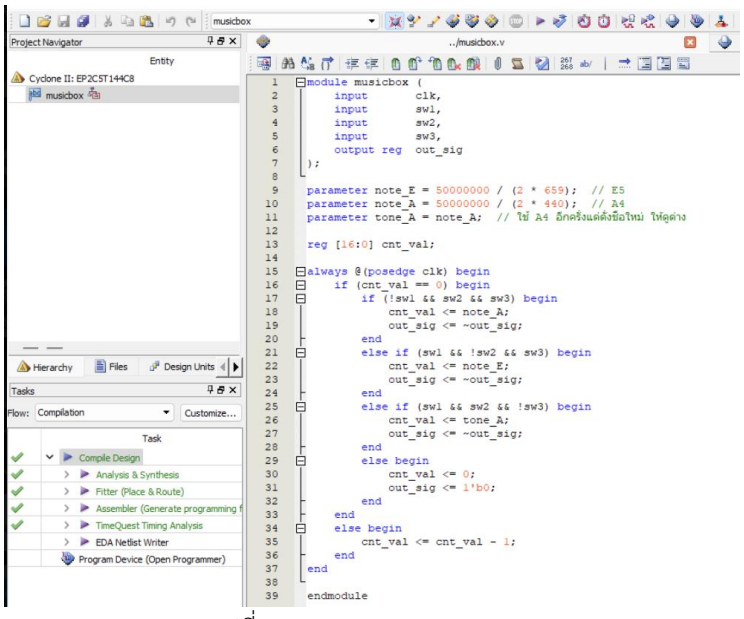
ชื่อ-นามสกุล ชวาทิก ธรฤทธิ์ รหัส 6610110066 Section 02

รายงานผลการทดลอง : 3HA04: Musicbox FPGA

ผลการทดลองCheckpoint #1 (รหัสนักศึกษา 3 ตัวสุดท้าย = 066 E5 A4 A4)

โมดูลนี้ชื่อว่า musicbox ทำหน้าที่สร้างสัญญาณ square waveออกมาจากการกดปุ่มสวิตช์ (sw1, sw2, sw3) เพื่อ ผ่านขา out_sig



```
module musicbox (
    input       clk,
    input       sw1,
    input       sw2,
    input       sw3,
    output reg  out_sig
);

parameter note_E = 50000000 / (2 * 659);  // E5
parameter note_A = 50000000 / (2 * 440);  // A4
parameter tone_A = note_A;  // ใช้ A4 อีกครั้งแต่ตั้งชื่อใหม่ ให้ดูต่าง

reg [16:0] cnt_val;

always @(posedge clk) begin
    if (cnt_val == 0) begin
        if (!sw1 && sw2 && sw3) begin
            cnt_val <= note_A;
            out_sig <= ~out_sig;
        end
        else if (sw1 && !sw2 && sw3) begin
            cnt_val <= note_E;
            out_sig <= ~out_sig;
        end
        else if (sw1 && sw2 && !sw3) begin
            cnt_val <= tone_A;
            out_sig <= ~out_sig;
        end
        else begin
            cnt_val <= 0;
            out_sig <= 1'b0;
        end
    end
    else begin
        cnt_val <= cnt_val - 1;
    end
end

endmodule
```

รูปที่ 1 code muscibox CP.1

1. Clock: ทำงานตามขอบขาขึ้น (posedge clk)

2. Counter (cnt_val) จะนับจำนวนรอบ clock

3. ถ้า counter ถึงค่าที่กำหนดตาม note ⟶ จะกลับค่า out_sig (toggle) เพื่อสร้างคลื่นสี่เหลี่ยม.

4. การเลือกโน้ตขึ้นอยู่กับการกดสวิตช์:

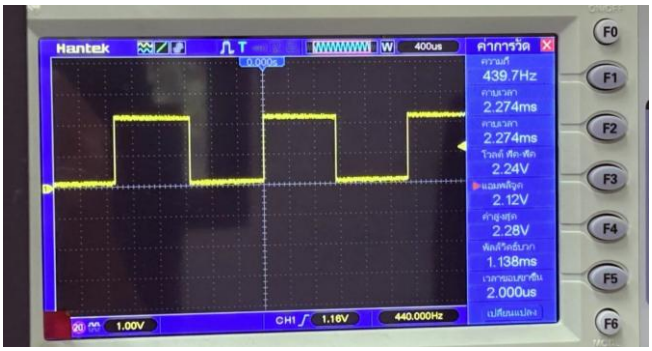
- sw1 ⟶ โน้ต E5
- sw2 & sw3 ⟶ โน้ต A4
- ไม่กดเลย ⟶ reset counterและ out_sig = 1

$$N = \frac{f_{clk}}{2 \times f_{note}}$$

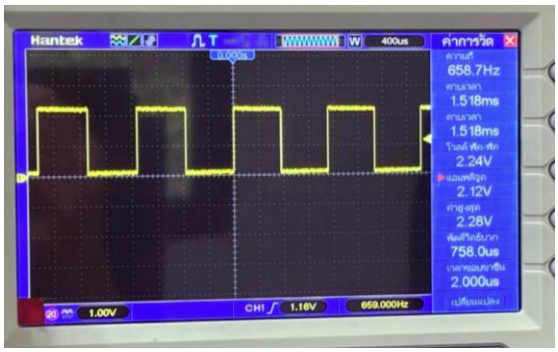
- parameter note_E = 50000000/(659*2); ⟶ ความถี่ ~659 Hz (โน้ต E5)
- parameter note_A = 50000000/(440*2); ⟶ ความถี่ ~440 Hz (โน้ต A4)
- cnt_val ขนาด 16 บิตเก็บการนับ clock

* เนื่องจากไม่ได้ทำการถ่าย Pin planner ไว้ จึงจะทำเป็น Table ให้แทน

Pin planner	
Clk	17
Sw1 (S5)	125
Sw2 (S6)	139
Sw3 (S7)	141



รูปที่ 2 signal note A4



รูปที่ 3 signal note E5

ชื่อ-นามสกุล ชวาทิก ธรฤทธิ์ รหัส 6610110066 Section 02

ผลการทดลองCheckpoint #2 (รหัสนักศึกษา 3 ตัวสุดท้าย = 066 E5 A4 A6) ส่งเสียงออกทาง buzzer ทั้ง3ตัว

```
parameter G4 = 50000000/392/2;
parameter A4 = 50000000/440/2;
parameter B4 = 50000000/493.88/2;
parameter C5 = 50000000/523.25/2;
parameter D5 = 50000000/587.33/2;
parameter E5 = 50000000/659.25/2;
parameter F5 = 50000000/698.46/2;
parameter G5 = 50000000/783.99/2;
parameter A5 = 50000000/880/2;
parameter B5 = 50000000/987.77/2;
parameter C6 = 50000000/1046.50/2;
parameter D6 = 50000000/1174.66/2;
parameter E6 = 50000000/1318.5/2;
parameter F6 = 50000000/1396.92/2;
parameter G6 = 50000000/1567.98/2;
parameter A6 = 50000000/1760/2;
parameter B6 = 50000000/1975.54/2;

reg [25:0] counter1, counter2, counter3;
reg spk1_out, spk2_out, spk3_out;
assign spk1 = spk1_out;
assign spk2 = spk2_out;
assign spk3 = spk3_out;
assign outF = clk;

// Logic for spk1
always @(posedge clk) begin
    if (sw1 == 0) begin
        if (counter1 == 0) begin
            spk1_out <= ~spk1_out;
            if (updown == 0) begin
                counter1 <= E5 - 1;
            end else begin
                counter1 <= E6 - 1;
            end
        end else begin
            counter1 <= counter1 - 1;
        end
    end else begin
        spk1_out <= 0;
        counter1 <= 0;
    end
```

รูปที่ 5 code Speaker 1

โมดูล musicbox สร้างเสียงจาก 3 ปุ่มกด (sw1, sw2, sw3) ออกที่ 3 buzzer (spk1, spk2, spk3)

มีสวิตช์ 1 บิตชื่อ updown ที่ เลือกระดับเสียง (octave):

- updown = 0 ⟶ เล่นชุดโน้ต “ต่ำ”: E5, A4, A5
- updown = 1 ⟶ ยกขึ้น “สูง” อีกหนึ่งระดับ: E6, A5, A6

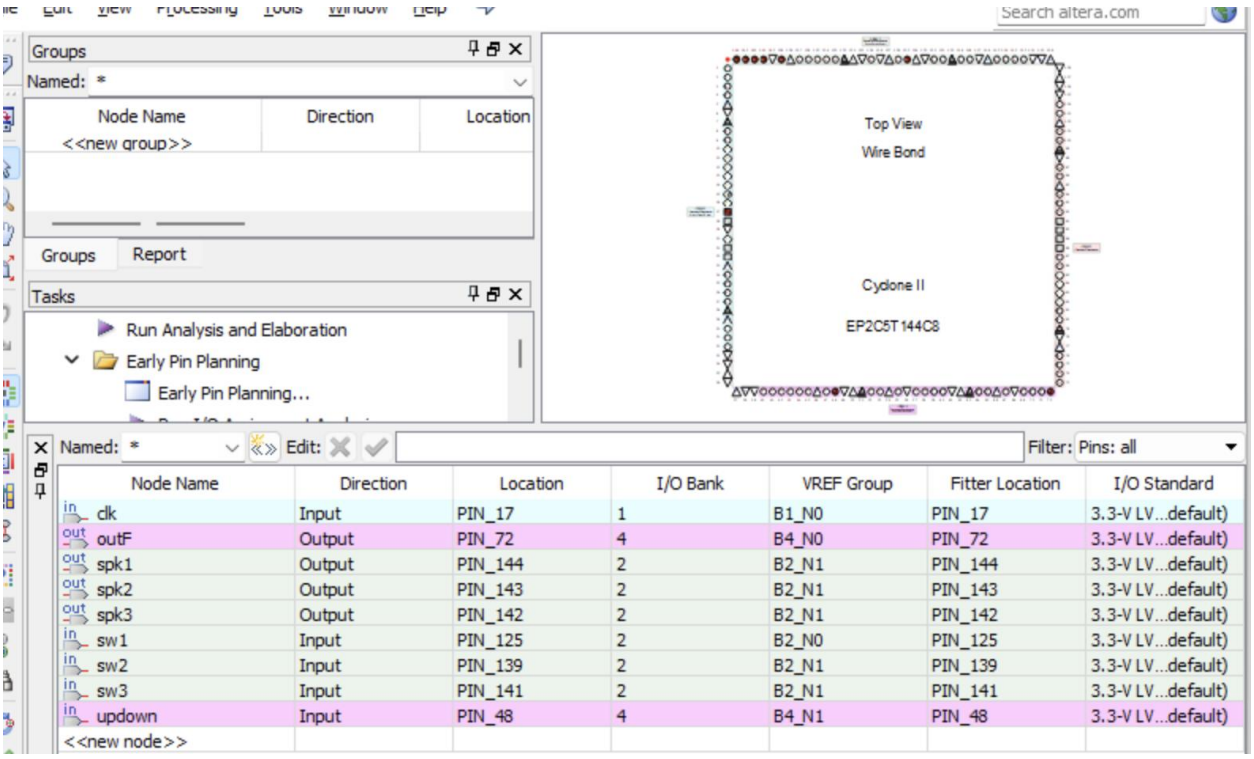
```
    // Logic for spk2
    always @(posedge clk) begin
        if (sw2 == 0) begin
            if (counter2 == 0) begin
                spk2_out <= ~spk2_out;
                if (updown == 0) begin
                    counter2 <= A4 - 1;
                end else begin
                    counter2 <= A5 - 1;
                end
            end else begin
                counter2 <= counter2 - 1;
            end
        end else begin
            spk2_out <= 0;
            counter2 <= 0;
        end
    end

    // Logic for spk3
    always @(posedge clk) begin
        if (sw3 == 0) begin
            if (counter3 == 0) begin
                spk3_out <= ~spk3_out;
                if (updown == 0) begin
                    counter3 <= A5 - 1;
                end else begin
                    counter3 <= A6 - 1;
                end
            end else begin
                counter3 <= counter3 - 1;
            end
        end else begin
            spk3_out <= 0;
            counter3 <= 0;
        end
    end
endmodule
```

รูปที่ 4 note + code Speaker 1และ2

1. **ลำโพง spk1 (เล่นโน้ต E5 หรือ E6)**
    - ถ้ากด sw1 ⟶ จะเล่นเสียง E5 (659 Hz) ถ้า updown=0 หรือ E6 (1319 Hz) ถ้า updown=1
    - ถ้าไม่กดสวิตช์ ⟶ เอาต์พุตเป็น 0 (ไม่มีเสียง)
2. **ลำโพง spk2 (เล่นโน้ต A4 หรือ A5)**
    - ถ้ากด sw2 ⟶ จะเล่นเสียง **A4 (440 Hz)** หรือ **A5 (880 Hz)** ตามค่า **updown**
3. **ลำโพง spk3 (เล่นโน้ต A5 หรือ A6)**
    - ถ้ากด sw3 ⟶ จะเล่นเสียง **A5 (880 Hz)** หรือ **A6 (1760 Hz)** ตามค่า **updown**
4. **หลักการทำงานโดยรวม**
    - เมื่อกดสวิตช์ ⟶ ตัวนับ (counter) จะเริ่มทำงาน
    - ตัวนับนับถอยหลังจนเหลือศูนย์ ⟶ toggle ค่าเอาต์พุต (spkX_out)
    - การ toggle ทำให้เกิด Square wave ที่ความถี่ตามโน้ต
    - ใช้สวิตช์ updown ในการสลับอ็อกเทฟ (ต่ำ/สูง)
    - ถ้าไม่กดสวิตช์ ⟶ ตัวนับและเอาต์พุตถูกรีเซ็ต (ไม่มีเสียง)
    - สามารถกดหลายสวิตช์พร้อมกันได้ ⟶ ทำให้ได้หลายเสียง (polyphony)

เสียง	สัญลักษณ์	ความถี่	สัญลักษณ์	ความถี่	สัญลักษณ์	ความถี่
“Do”	C4	261.63	C5	523.25	C6	1046.50
“Re”	D4	293.66	D5	587.33	D6	1174.66
“Mi”	E4	329.63	E5	659.25	E6	1318.50
“Fa”	F4	349.23	F5	698.46	F6	1396.92
“Sol”	G4	392.00	G5	783.99	G6	1567.98
“La”	A4	440.00	A5	880.00	A6	1760.00
“Si”	B4	493.88	B5	987.77	B6	1975.54

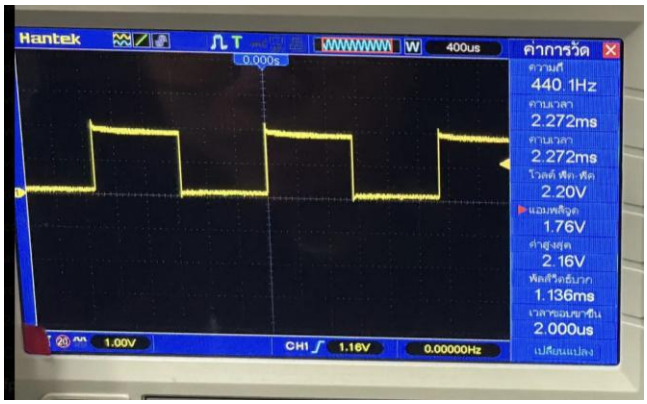
รูปที่ 6 ตาราง note และความถี่



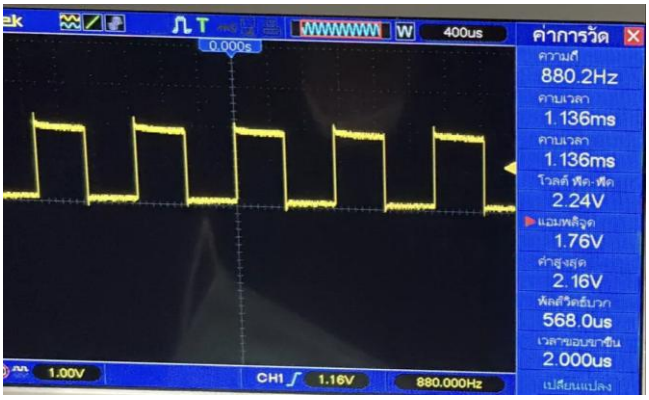
รูปที่ 7 Pin Planner

Pin planner	
Clk	17
OutF เวลากด	72
Spk 1	144
Spk 2	143
Spk 3	142
Sw1 (S5)	125
Sw2 (S6)	139
Sw3 (S7)	141
Updown DIP (S4)	48 (active low)

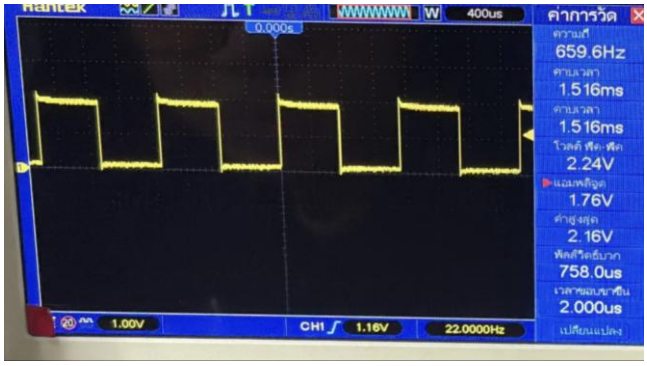
ชื่อ-นามสกุล ชวาทิก ธรฤทธิ์ รหัส 6610110066 Section 02



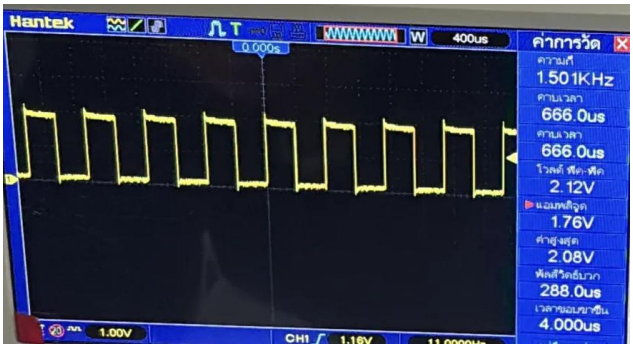
รูปที่ 8 note La A4



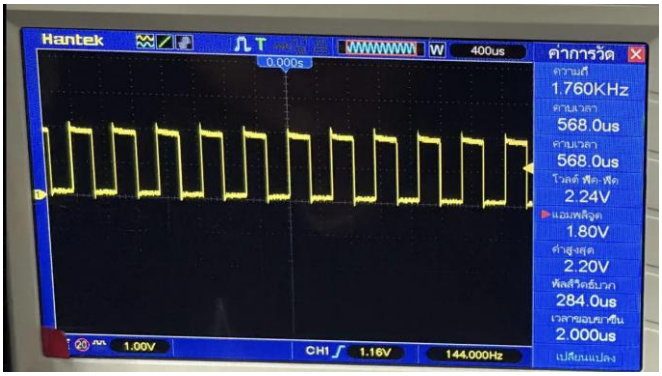
รูปที่ 9 +1 octave note La A5



รูปที่ 10 note Mi E5



รูปที่ 11 +1 octave note Mi E6 (เพี้ยน)



รูปที่ 12 note La A6

ชื่อ-นามสกุล ชวาทิก ธรฤทธิ์ รหัส 6610110066 Section 02

Checkpoint#3 ให้ทำการสร้างเพลงเอง

เพลงที่เลือก: Kamado Tanjiro no Uta จาก demon slayer

*แต่เนื่องจาก code ที่ทำไม่มี duration ของแต่ละ note จังหวะของเสียงทำให้การฟังจึงไม่เหมือนนัก แต่โน้ตคือโน้ตเดียวกัน

```
reg [31:0] note_div   [0:MELODY_LEN-1];
reg [7:0]  note_dur   [0:MELODY_LEN-1];

initial begin
    integer i;
    for (i=0;i<MELODY_LEN;i=i+1) begin
        note_div[i] = REST; note_dur[i] = 1;
    end

    // ---- Phrase A (D5 C5 D5 | D5 F5 A5 | F5 E5 | C5 F5 A5) ----
    note_div[0]  = D5;  note_dur[0]  = 2;
    note_div[1]  = C5;  note_dur[1]  = 1;
    note_div[2]  = D5;  note_dur[2]  = 1;

    note_div[3]  = D5;  note_dur[3]  = 2;
    note_div[4]  = F5;  note_dur[4]  = 1;
    note_div[5]  = A5;  note_dur[5]  = 1;

    note_div[6]  = F5;  note_dur[6]  = 2;
    note_div[7]  = E5;  note_dur[7]  = 2;

    note_div[8]  = C5;  note_dur[8]  = 2;
    note_div[9]  = F5;  note_dur[9]  = 1;
    note_div[10] = A5;  note_dur[10] = 1;

    // ---- Phrase B (D5 D5 F5 | F5 A5 C6 | A5 G5 | A5 G5 D6) ----
    note_div[11] = D5;  note_dur[11] = 2;
    note_div[12] = D5;  note_dur[12] = 2;

    note_div[13] = F5;  note_dur[13] = 2;
    note_div[14] = F5;  note_dur[14] = 1;
    note_div[15] = A5;  note_dur[15] = 1;

    note_div[16] = C6;  note_dur[16] = 2;
    note_div[17] = A5;  note_dur[17] = 2;

    note_div[18] = G5;  note_dur[18] = 2;
    note_div[19] = A5;  note_dur[19] = 1;
    note_div[20] = G5;  note_dur[20] = 1;

    note_div[21] = D6;  note_dur[21] = 4; // hold
```

```
    // ---- Bridge (C6 A5 G5 | F5 E5 D5 | REST | pickup) ----
    note_div[22] = C6;  note_dur[22] = 2;
    note_div[23] = A5;  note_dur[23] = 2;

    note_div[24] = G5;  note_dur[24] = 2;
    note_div[25] = F5;  note_dur[25] = 1;
    note_div[26] = E5;  note_dur[26] = 1;

    note_div[27] = D5;  note_dur[27] = 4;

    note_div[28] = REST;note_dur[28] = 2;
    note_div[29] = D5;  note_dur[29] = 2; // pickup to reprise

    // ---- Phrase A' (variant) ----
    note_div[30] = D5;  note_dur[30] = 2;
    note_div[31] = C5;  note_dur[31] = 1;
    note_div[32] = D5;  note_dur[32] = 1;

    note_div[33] = F5;  note_dur[33] = 2;
    note_div[34] = A5;  note_dur[34] = 2;

    note_div[35] = F5;  note_dur[35] = 2;
    note_div[36] = E5;  note_dur[36] = 2;

    note_div[37] = C5;  note_dur[37] = 2;
    note_div[38] = F5;  note_dur[38] = 1;
    note_div[39] = A5;  note_dur[39] = 1;

    // ---- Cadence ----
    note_div[40] = D5;  note_dur[40] = 2;
    note_div[41] = D6;  note_dur[41] = 2;

    note_div[42] = C6;  note_dur[42] = 2;
    note_div[43] = A5;  note_dur[43] = 2;

    note_div[44] = G5;  note_dur[44] = 2;
    note_div[45] = F5;  note_dur[45] = 1;
    note_div[46] = E5;  note_dur[46] = 1;

    note_div[47] = D5;  note_dur[47] = 4; // end/loop point
end

// ===== Tone divider =====
reg [31:0] tone_cnt;
wire       tone_tick = (tone_cnt == 0);
```

```
module musicbox_tanjiro (
    input        clk,          // 50 MHz
    input        rst_n,        // active-low reset
    input        play,         // 1 = play/loop, 0 = idle
    output reg   out_sig       // square-wave audio out
);

// ===== User tunables =====
localparam integer CLK_HZ      = 50_000_000;
localparam integer BPM_DEFAULT = 99;            // ~official tempo
localparam integer NOTE_UNIT   = 8;             // durations are in 1/8 notes

// ===== Tempo tick generator (1 "duration unit" = 1/8 note) =====
// duration_tick_hz = BPM * NOTE_UNIT / 60
localparam integer DUR_TICK_HZ = (BPM_DEFAULT * NOTE_UNIT) / 60;
localparam integer DUR_DIV     = CLK_HZ / DUR_TICK_HZ;  // cycles per 1/8 note

reg [31:0] dur_cnt;
wire       dur_tick = (dur_cnt == 0);

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) dur_cnt <= DUR_DIV - 1;
    else if (!play) dur_cnt <= DUR_DIV - 1;
    else dur_cnt <= dur_tick ? (DUR_DIV - 1) : (dur_cnt - 1);
end

// ===== Note frequency to divider (period/2 in clk cycles) =====
// period_half = CLK_HZ / (2*freq)
// Define required pitches around D minor (D4..D6 range)
localparam integer C4  = CLK_HZ/(2*262);  localparam integer Cs4 = CLK_HZ/(2*277);
localparam integer D4  = CLK_HZ/(2*294);  localparam integer Ds4 = CLK_HZ/(2*311);
localparam integer E4  = CLK_HZ/(2*330);  localparam integer F4  = CLK_HZ/(2*349);
localparam integer Fs4 = CLK_HZ/(2*370);  localparam integer G4  = CLK_HZ/(2*392);
localparam integer Gs4 = CLK_HZ/(2*415);  localparam integer A4  = CLK_HZ/(2*440);
localparam integer As4 = CLK_HZ/(2*466);  localparam integer B4  = CLK_HZ/(2*494);

localparam integer C5  = CLK_HZ/(2*523);  localparam integer Cs5 = CLK_HZ/(2*554);
localparam integer D5  = CLK_HZ/(2*587);  localparam integer Ds5 = CLK_HZ/(2*622);
localparam integer E5  = CLK_HZ/(2*659);  localparam integer F5  = CLK_HZ/(2*698);
localparam integer Fs5 = CLK_HZ/(2*740);  localparam integer G5  = CLK_HZ/(2*784);
localparam integer Gs5 = CLK_HZ/(2*831);  localparam integer A5  = CLK_HZ/(2*880);
localparam integer As5 = CLK_HZ/(2*932);  localparam integer B5  = CLK_HZ/(2*988);

localparam integer C6  = CLK_HZ/(2*1046); localparam integer Cs6 = CLK_HZ/(2*1109);
localparam integer D6  = CLK_HZ/(2*1175); localparam integer Ds6 = CLK_HZ/(2*1245);
localparam integer E6  = CLK_HZ/(2*1319); localparam integer F6  = CLK_HZ/(2*1397);
localparam integer Fs6 = CLK_HZ/(2*1480); localparam integer G6  = CLK_HZ/(2*1568);
```

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        tone_cnt <= 0;
        out_sig  <= 1'b0;
    end else if (!play || current_div == REST) begin
        tone_cnt <= 0;
        out_sig  <= 1'b0;
    end else begin
        if (tone_tick) begin
            tone_cnt <= current_div - 1;
            out_sig  <= ~out_sig;
        end else begin
            tone_cnt <= tone_cnt - 1;
        end
    end
end

// ===== Sequencer (indexes ROM, counts durations) =====
reg [7:0]  idx;
reg [15:0] dur_left;        // in 1/8-note ticks
reg [31:0] current_div;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        idx         <= 0;
        current_div <= REST;
        dur_left    <= 0;
    end else if (!play) begin
        idx         <= 0;
        current_div <= REST;
        dur_left    <= 0;
    end else begin
        // Load first/next note when needed
        if (dur_left == 0) begin
            current_div <= note_div[idx];
            dur_left    <= note_dur[idx];         // load duration (in 1/8 notes)
        end else if (dur_tick) begin
            dur_left <= dur_left - 1;
            if (dur_left == 1) begin
                // advance index on next tick
                if (idx == MELODY_LEN-1)
                    idx <= 0;                     // loop
                else
                    idx <= idx + 1;
            end
        end
    end
end
```

ชื่อ-นามสกุล ชวาทิก ธรฤทธิ์ รหัส 6610110066 Section 02

1. ภาพรวม (Overview)
    - สร้างเสียงดนตรีออกที่ขา out_sig โดยใช้สัญญาณนาฬิกา clk 50 MHz
    - เมื่อ play=1 ⟶ เริ่มเล่นทำนองและวนลูปอัตโนมัติ
    - เมื่อ play=0 ⟶ หยุดเล่น (out = เงียบ)
    - เก็บข้อมูลโน้ต (frequency divider) และความยาวโน้ต (duration) ของทำนอง

2. Input / Output
    - Input
        - clk : สัญญาณนาฬิกาหลัก 50 MHz
        - rst_n : reset แบบ active-low
        - play : สวิตช์เริ่ม/หยุดเพลง
    - Output
        - out_sig : สัญญาณ square wave ที่ส่งไปขับ buzzer
3. Tempo Generator
    - localparam BPM_DEFAULT = 99; // BPM 99/นาที
    - localparam NOTE_UNIT = 8; // หน่วยเป็นโน้ตเขบ็ดหนึ่งชั้น (1/8)
    - มีตัวนับ dur_cnt ใช้นับ cycle ของ clock จนได้จังหวะ 1/8 note
4. Note Frequency Table
    - เก็บค่าครึ่งคาบ (period/2) สำหรับโน้ต C4–G6
5. Melody ROM (เก็บทำนอง)

    note_div[0] = D5; note_dur[0] = 2; // โน้ต D5 ความยาว 1/4

    note_div[1] = C5; note_dur[1] = 1; // โน้ต C5 ความยาว 1/8

    - note_div[i] = ค่าความถี่ (divider) ของโน้ต
    - note_dur[i] = ระยะเวลา (เป็นจำนวน 1/8 note)
6. Tone Divider (สร้างเสียงจริง)
    - reg [31:0] tone_cnt : **tone_cnt** จะนับลงจากค่า **current_div**
    - wire tone_tick = (tone_cnt == 0) : ทุกครั้งที่ถึง 0 ⟶ toggle **out_sig** ⟶ เกิด Square wave

7. **Sequencer (ตัวจัดลำดับเล่นทำนอง)**
    - ทำหน้าที่เลือกโน้ตจาก Melody ROM
    - โหลด note_div และ note_dur เมื่อเริ่มโน้ตใหม่
    - ลดค่า dur_left ทุก ๆ duration tick
    - เมื่อหมด ⟶ ไปยังโน้ตถัดไป (idx+1)
    - ถ้าเล่นถึงโน้ตสุดท้าย ⟶ กลับไปที่ index 0 (วนซ้ำ)
8. **หลักการทำงานโดยรวม**
    - กดปุ่ม play=1 ⟶ Melody ROM จะเริ่มทำงาน
    - Sequencer จะเลือกโน้ตตัวแรก ⟶ โหลดค่า divider และ duration
    - Tone divider ใช้ divider ในการแบ่ง clock ออกมาเป็นความถี่เสียงจริง
    - Tempo generator จับเวลาเพื่อกำหนดว่าโน้ตจะเล่นนานแค่ไหน
    - เมื่อโน้ตหมดเวลา ⟶ เปลี่ยนไปโน้ตถัดไป
    - ทำซ้ำไปเรื่อย ๆ จนครบทำนอง แล้ววนกลับไปต้นเพลงอีกครั้ง
9. **YouTube Vdo Link**: https://youtube.com/shorts/6Jnp6_HoiN8?si=o16xMznAG56mtFTx