Welcome to The Logic Design Lab!

Fall 2021
Audio Tutorial

Prof. Chun-Yi Lee

Department of Computer Science
National Tsing Hua University

Agenda

- Announcements
- Keyboard Module: SampleDisplay
- Audio Tutorial
  - Concepts
  - Demo 1
  - Demo 2

Today's class will help you:

1. Understand how to revise the provided SampleDisplay module
2. Understand the concept of the Audio module
3. Understand how to revise the Audio module

Announcements

- Lab 5
  - Lab 5 demonstration on **11/25/2020 (Thu)**
  - **Please read the lab description and specification carefully**
  - **Compile your codes again before merging yours with your teammates**
  - **Please follow the template I/Os and submit your .v files**
- Final project
  - You can begin working on your final projects from now
  - Please avoid copying the works from past years

Final Project

- Use your FPGA to implement creative and interesting works

- Final project proposal due on **12/13/2021 (Mon)**
- Final project report due on **1/14/2022 (Fri)**

- Final exam
  - **12/9/2021 (Thu)**
  - 2 hours

Final Project Presentation

- Final project demonstration on **1/13/2022 (Thu)** and **1/14/2022 (Fri)**
  - Around 10 minutes per team

- **No restriction on your presentation style**
  - Be creative!
  - What is special and new in your project
  - Key features

- Order of presentation
  - We will randomly decide the order
  - Let us know if you have constraints

Final Project Report

- Final project report
  - Due on **1/14/2022, 23:59pm (Fri)**
  - Block diagrams and state transition diagrams
  - **Detailed explanation**

- Report contents
  - Introduction
  - Motivation
  - System specification
  - Experimental results
  - Conclusion
  - …And any other sections that you would like to include

Final Project Award

- **Category:** Difficulty and completeness
  - Graded by me and the TAs
    - Difficulty (**35%**)
    - Completeness (**30%**)
    - Source code coding style (**15%**) (Correctness, usage, comments, etc.)
    - Report (**20%**)
  - First place: **NTD $6,000**
  - Second place: **NTD $3,000**
  - Third place: **NTD $1,500**

- **Category:** Best creativity
  - Voted by everyone in the class
  - **NTD $1,000**

- **Category:** Deep learning models in FPGA
  - Extra **bonus points** and **cash rewards**. To be announced

Final Exam

- Dates and time
  - **12/9/2021 (Thu)**
  - **3:30pm ~ 5:30pm (2 hours)**

- Course contents
  - Verilog design questions
  - Logic design concepts
  - Lecture & lab contents

Agenda

- Announcements
- Keyboard Module: SampleDisplay
- Audio Tutorial
  - Concepts
  - Demo 1
  - Demo 2

Today's class will help you:

1. Understand how to revise the provided SampleDisplay module
2. Understand the concept of the Audio module
3. Understand how to revise the Audio module

Verilog Module: SampleDisplay

- In Keyboard Sample Code
  - KeyboardDecoder.v
  - SampleDisplay.v

- I/O for KeyboardDecoder

Output

- display
- digit

Input

- PS2_CLK
- PS2_DATA
- rst
- clk

Verilog Module: SampleDisplay

- In SampleDisplay module, you can define the parameters for the make codes of the keys (9 bits)
- The first bit stands for Extended Bit (1: Yes, 0: No), while the last eight bits stand for the make codes

```
parameter [8:0] LEFT_SHIFT_CODES  = 9'b0_0001_0010;
parameter [8:0] RIGHT_SHIFT_CODES = 9'b0_0101_1001;
parameter [8:0] KEY_CODES [0:19] = {
    9'b0_0100_0101, // 0 => 45
    9'b0_0001_0110, // 1 => 16
    9'b0_0001_1110, // 2 => 1E
    9'b0_0010_0110, // 3 => 26
    9'b0_0010_0101, // 4 => 25
    9'b0_0010_1110, // 5 => 2E
    9'b0_0011_0110, // 6 => 36
    9'b0_0011_1101, // 7 => 3D
    9'b0_0011_1110, // 8 => 3E
    9'b0_0100_0110, // 9 => 46

    9'b0_0111_0000, // right_0 => 70
    9'b0_0110_1001, // right_1 => 69
    9'b0_0111_0010, // right_2 => 72
    9'b0_0111_1010, // right_3 => 7A
    9'b0_0110_1011, // right_4 => 6B
    9'b0_0111_0011, // right_5 => 73
    9'b0_0111_0100, // right_6 => 74
    9'b0_0110_1100, // right_7 => 6C
    9'b0_0111_0101, // right_8 => 75
    9'b0_0111_1101  // right_9 => 7D
};
```

Verilog Module: SampleDisplay

- Use **keydown** to track which key is pressed
- Use **lastchange** to track which key is just hit
  - Use **been_ready** and **clk** to trigger the key-pressing event

```
assign shift_down = (key_down[LEFT_SHIFT_CODES] == 1'b1 || key_down[RIGHT_SHIFT_CODES] == 1'b1) ? 1'b1 : 1'b0;
```

```
always @ (posedge clk, posedge rst) begin
    if (rst) begin
        nums <= 16'b0;
    end else begin
        nums <= nums;
        if (been_ready && key_down[last_change] == 1'b1) begin
            if (key_num != 4'b1111)begin
                if (shift_down == 1'b1) begin
                    nums <= {key_num, nums[15:4]};
                end else begin
                    nums <= {nums[11:0], key_num};
                end
            end
        end
    end
end
```

Agenda

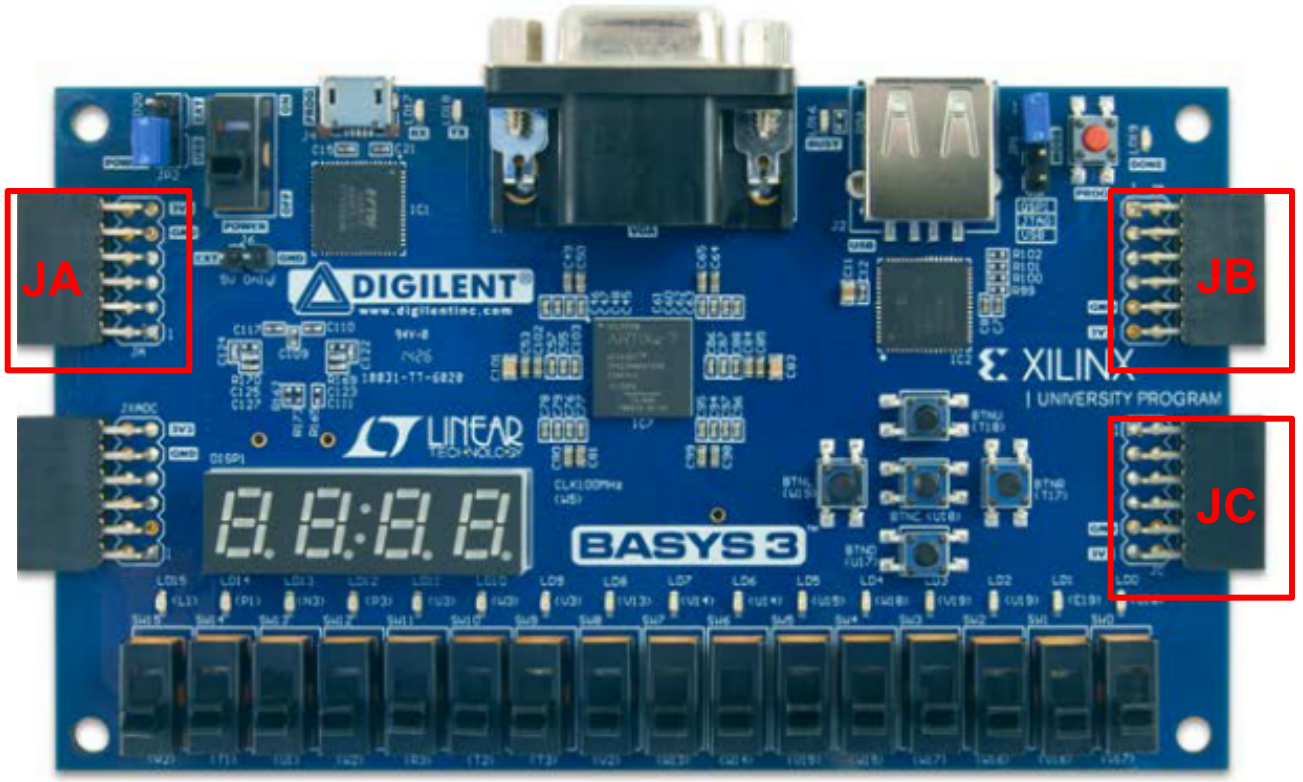
- Announcements
- Keyboard Module: SampleDisplay
- Audio Tutorial
  - Concepts
  - Demo 1
  - Demo 2

Today's class will help you:

1. Understand how to revise the provided SampleDisplay module
2. Understand the concept of the Audio module
3. Understand how to revise the Audio module

Pmod Connectors (1/2)

- Basys3 provides 3 Pmod connectors
  - JA, JB, and JC



Pmod Connectors (2/2)

- Each 12-pin Pmod connector provides
  - **Two** 3.3V VCC signals (pins 6 and 12)
  - **Two** Ground signals (pins 5 and 11)
  - **Eight** logic signals

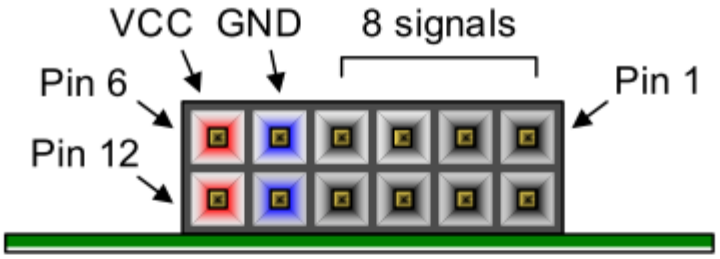
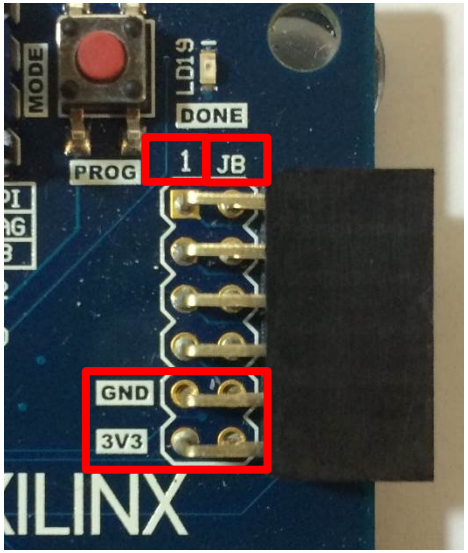


Figure 20. Pmod connectors; front view as loaded on PCB.



Pin Assignments

- Corresponding pins in a constraint file (.xdc)

```
##Pmod Header JA
##Sch name = JA1
#set_property PACKAGE_PIN J1 [get_ports {JA[0]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {JA[0]}]
##Sch name = JA2
#set_property PACKAGE_PIN L2 [get_ports {JA[1]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {JA[1]}]
##Sch name = JA3
#set_property PACKAGE_PIN J2 [get_ports {JA[2]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {JA[2]}]
##Sch name = JA4
#set_property PACKAGE_PIN G2 [get_ports {JA[3]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {JA[3]}]
##Sch name = JA7
#set_property PACKAGE_PIN H1 [get_ports {JA[4]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {JA[4]}]
##Sch name = JA8
#set_property PACKAGE_PIN K2 [get_ports {JA[5]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {JA[5]}]
##Sch name = JA9
#set_property PACKAGE_PIN H2 [get_ports {JA[6]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {JA[6]}]
##Sch name = JA10
#set_property PACKAGE_PIN G3 [get_ports {JA[7]}]
#set_property IOSTANDARD LVCMOS33 [get_ports {JA[7]}]
```

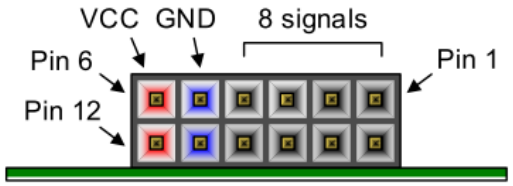


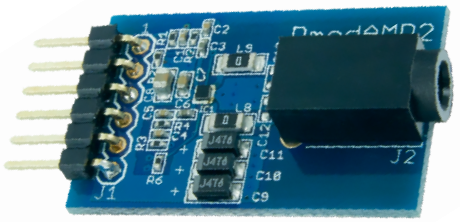
Figure 20. Pmod connectors; front view as loaded on PCB.

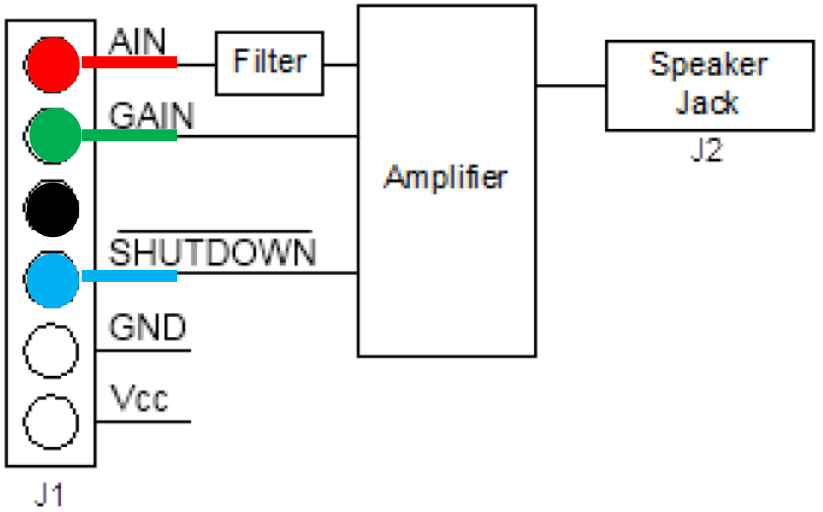
Pmod JA	Pmod JB	Pmod JC
JA1: J1	JB1: A14	JC1: K17
JA2: L2	JB2: A16	JC2: M18
JA3: J2	JB3: B15	JC3: N17
JA4: G2	JB4: B16	JC4: P18
JA7: H1	JB7: A15	JC7: L17
JA8: K2	JB8: A17	JC8: M19
JA9: H2	JB9: C15	JC9: P17
JA10: G3	JB10: C16	JC10: R18

Table 6. Basys3 Pmod pin assignme

Pmod Accessory Board : "PmodAMP2"

- **AIN (audio_in)**
- **GAIN (default 1)**
  - 1'b1: 6dB
  - 1'b0: 12dB
- **SHUTDOWN (default 1)**
  - 1'b0: disable
  - 1'b1: enable





Ref. *"PmodAMP2 Reference Manual"*, revision: Aug 2, 2012, Digilent

Basic Concepts of Sound

- **Sound** that is perceptible by **human beings** has frequencies ranging from about **20** Hz to **20,000** Hz
- **Frequency**:
  - The **higher** (**lower**) the frequency, the **higher** (**lower**) the pitch
- **Duty cycle**:
  - The **evener** the duty cycle, the **better** the quality.

音高 (Pitch)

唱名	Do-H	Re-H	Mi-H	Fa-H	So-H	La-H	Si-H
數字	· 1	· 2	· 3	· 4	· 5	· 6	· 7
頻率(Hz)	524	588	660	698	784	880	988

唱名	Do	Re	Mi	Fa	So	La	Si
音高	C	D	E	F	G	A	B
數字	1	2	3	4	5	6	7
頻率(Hz)	262	294	330	349	392	440	494

唱名	Do-L	Re-L	Mi-L	Fa-L	So-L	La-L	Si-L
數字	1 ·	2 ·	3 ·	4 ·	5 ·	6 ·	7 ·
頻率(Hz)	131	147	165	174	196	220	247

Ref. “音高”, wikipedia
Ref. “Numbered Musical Notation (簡譜)”, wikipedia

頻率，單位為赫茲（括號內為半音距離，"（0）"為中央C）										
C	16.352（−48）	32.703（−36）	65.406（−24）	130.81（−12）	261.63 (0)	523.25（+12）	1046.5（+24）	2093.0（+36）	4186.0（+48）	8372.0（+60）
C♯/D♭	17.324（−47）	34.648（−35）	69.296（−23）	138.59（−11）	277.18（+1）	554.37（+13）	1108.7（+25）	2217.5（+37）	4434.9（+49）	8869.8（+61）
D	18.354（−46）	36.708（−34）	73.416（−22）	146.83（−10）	293.66（+2）	587.33（+14）	1174.7（+26）	2349.3（+38）	4698.6（+50）	9397.3（+62）
D♯/E♭	19.445（−45）	38.891（−33）	77.782（−21）	155.56（−9）	311.13（+3）	622.25（+15）	1244.5（+27）	2489.0（+39）	4978.0（+51）	9956.1（+63）
E	20.602（−44）	41.203（−32）	82.407（−20）	164.81（−8）	329.63（+4）	659.26（+16）	1318.5（+28）	2637.0（+40）	5274.0（+52）	10548（+64）
F	21.827（−43）	43.654（−31）	87.307（−19）	174.61（−7）	349.23（+5）	698.46（+17）	1396.9（+29）	2793.8（+41）	5587.7（+53）	11175（+65）
F♯/G♭	23.125（−42）	46.249（−30）	92.499（−18）	185.00（−6）	369.99（+6）	739.99（+18）	1480.0（+30）	2960.0（+42）	5919.9（+54）	11840（+66）
G	24.500（−41）	48.999（−29）	97.999（−17）	196.00（−5）	392.00（+7）	783.99（+19）	1568.0（+31）	3136.0（+43）	6271.9（+55）	12544（+67）
G♯/A♭	25.957（−40）	51.913（−28）	103.83（−16）	207.65（−4）	415.30（+8）	830.61（+20）	1661.2（+32）	3322.4（+44）	6644.9（+56）	13290（+68）
A	27.500（−39）	55.000（−27）	110.00（−15）	220.00（−3）	440.00（+9）	880.00（+21）	1760.0（+33）	3520.0（+45）	7040.0（+57）	14080（+69）
A♯/B♭	29.135（−38）	58.270（−26）	116.54（−14）	233.08（−2）	466.16（+10）	932.33（+22）	1864.7（+34）	3729.3（+46）	7458.6（+58）	14917（+70）
B	30.868（−37）	61.735（−25）	123.47（−13）	246.94（−1）	493.88（+11）	987.77（+23）	1975.5（+35）	3951.1（+47）	7902.1（+59）	15804（+71）

Ref. “音高”, wikipedia

Agenda

- Announcements
- Keyboard Module: SampleDisplay
- Audio Tutorial
  - Concepts
  - Demo 1
  - Demo 2

Today's class will help you:

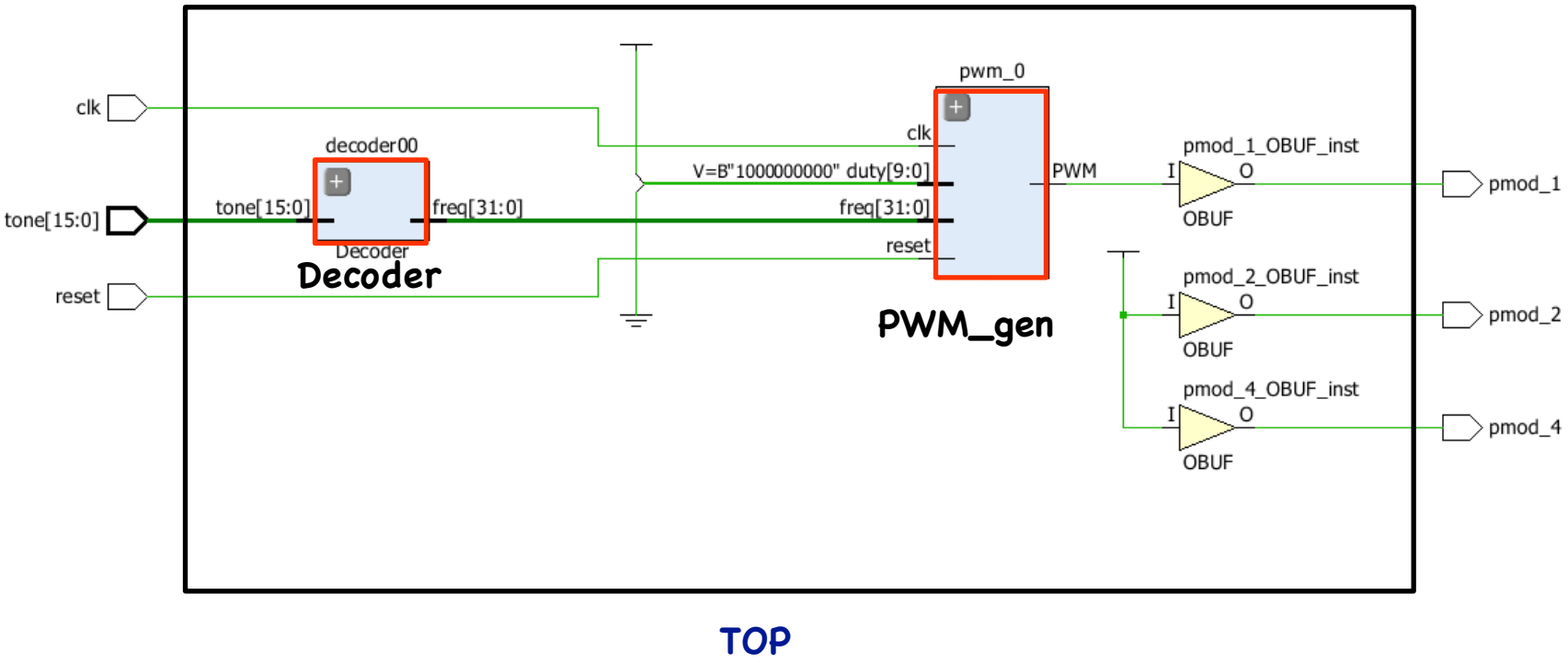
1. Understand how to revise the provided SampleDisplay module
2. Understand the concept of the Audio module
3. Understand how to revise the Audio module

Demo 1: Piano

- **Input**: press a key
- **Output**: make a matched sound

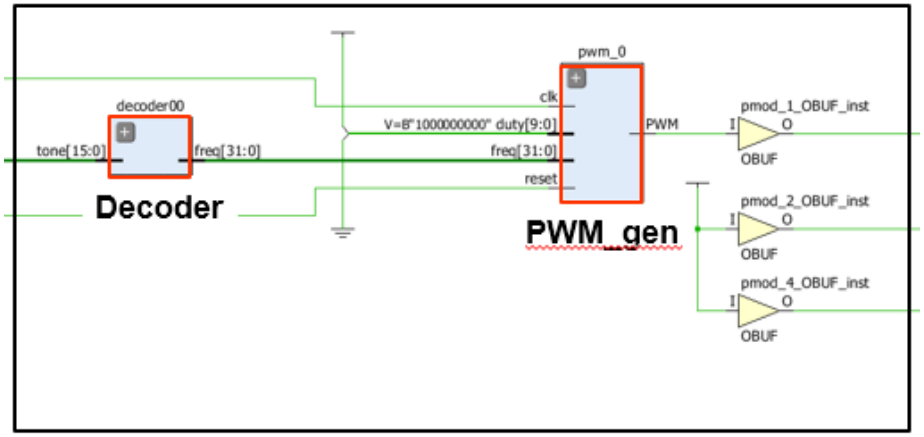
- Also can be used to test the PmodAMP2 board

Demo 1: Block Diagram



Demo 1: Top Module

- In the demo code, only pin1, pin2, and pin4 are used
  - Pin 1: AIN
  - Pin 2: GAIN
  - Pin 4: SHUTDOWN_N
- The top module includes two sub-modules
  - Decoder
  - PWN_gen



TOP

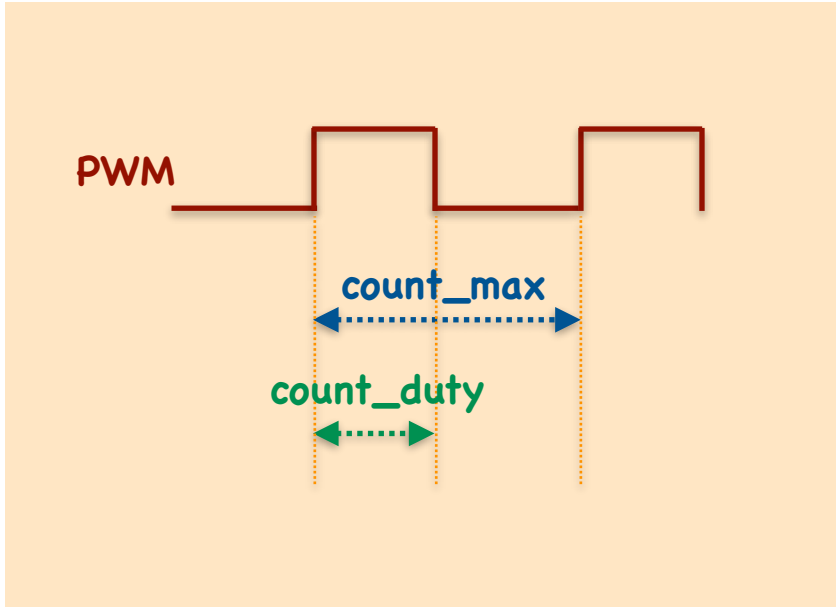
```
module TOP (
    input clk,
    input reset,
    input [15:0] tone, //piano keys
    output pmod_1,  //AIN
    output pmod_2,  //GAIN
    output pmod_4   //SHUTDOWN_N
);

wire [31:0] freq;
assign pmod_2 = 1'd1;   //no gain(6dB)
assign pmod_4 = 1'd1;   //turn-on

Decoder decoder00 (
    .tone(tone),
    .freq(freq)
);

PWM_gen pwm_0 (
    .clk(clk),
    .reset(reset),
    .freq(freq),
    .duty(10'd512),
    .PWM(pmod_1)
);

endmodule
```

Demo 1: Decoder

```
module Decoder (
    input [15:0] tone,
    output reg [31:0] freq
);

always @(*) begin
    case (tone)
        16'b0000_0000_0000_0001: freq = 32'd262;    //Do-m
        16'b0000_0000_0000_0010: freq = 32'd294;    //Re-m
        16'b0000_0000_0000_0100: freq = 32'd330;    //Mi-m
        16'b0000_0000_0000_1000: freq = 32'd349;    //Fa-m
        16'b0000_0000_0001_0000: freq = 32'd392;    //Sol-m
        16'b0000_0000_0010_0000: freq = 32'd440;    //La-m
        16'b0000_0000_0100_0000: freq = 32'd494;    //Si-m
        16'b0000_0000_1000_0000: freq = 32'd262 << 1;   //Do-h
        16'b0000_0001_0000_0000: freq = 32'd294 << 1;
        16'b0000_0010_0000_0000: freq = 32'd330 << 1;
        16'b0000_0100_0000_0000: freq = 32'd349 << 1;
        16'b0000_1000_0000_0000: freq = 32'd392 << 1;
        16'b0001_0000_0000_0000: freq = 32'd440 << 1;
        16'b0010_0000_0000_0000: freq = 32'd494 << 1;
        16'b0100_0000_0000_0000: freq = 32'd262 << 2;
        16'b1000_0000_0000_0000: freq = 32'd294 << 2;
        default : freq = 32'd20000; //Do-dummy
    endcase
end

endmodule
```

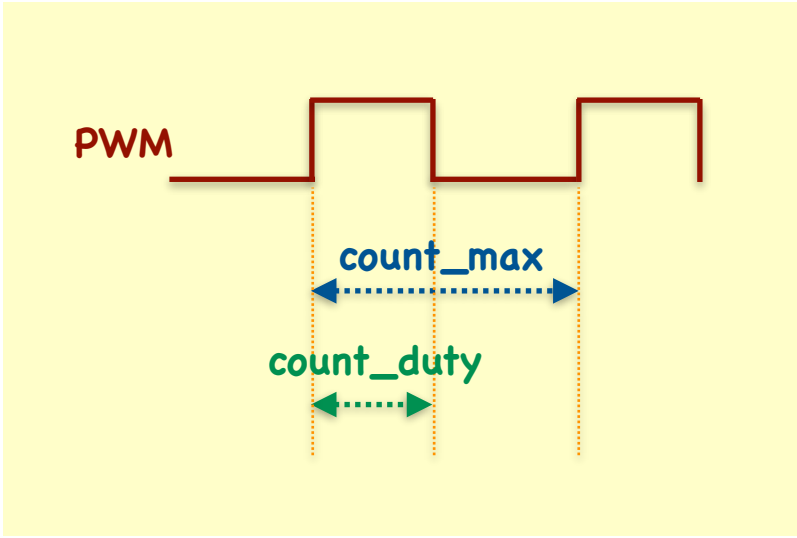
Demo 1: PWM (Pulse Width Modulation)

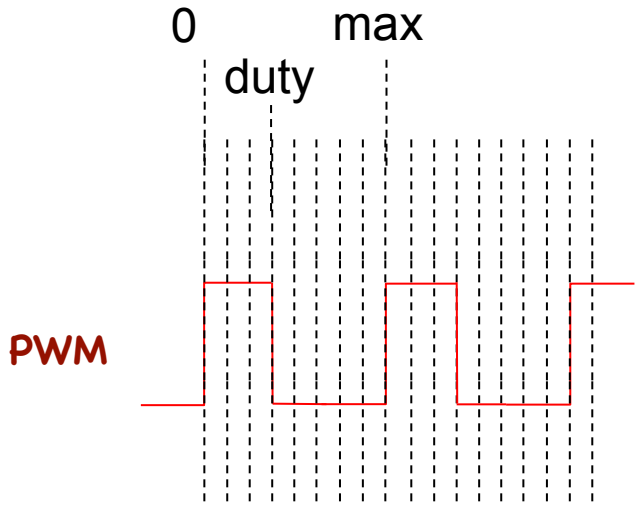
- This module expects 100 MHz input clock
  - **Input**: Frequency
  - **Input**: Desired duty cycle
- Generate repeated pulses with specified frequency (in Hz)
  - Should be less than or equal to 100 MHz
- Duty cycle
  - Vary in step of 1/1024 (approximately 0.1%)



Demo 1: PWM
(Pulse Width Modulation)

```
module PWM_gen (
    input wire clk,
    input wire reset,
    input [31:0] freq,
    input [9:0] duty,
    output reg PWM
);

wire [31:0] count_max = 100_000_000 / freq;
wire [31:0] count_duty = count_max * duty / 1024;
reg [31:0] count;

always @(posedge clk, posedge reset) begin
    if (reset) begin
        count <= 0;
        PWM <= 0;
    end else if (count < count_max) begin
        count <= count + 1;
        if(count < count_duty)
            PWM <= 1;
        else
            PWM <= 0;
    end else begin
        count <= 0;
        PWM <= 0;
    end
end

endmodule
```





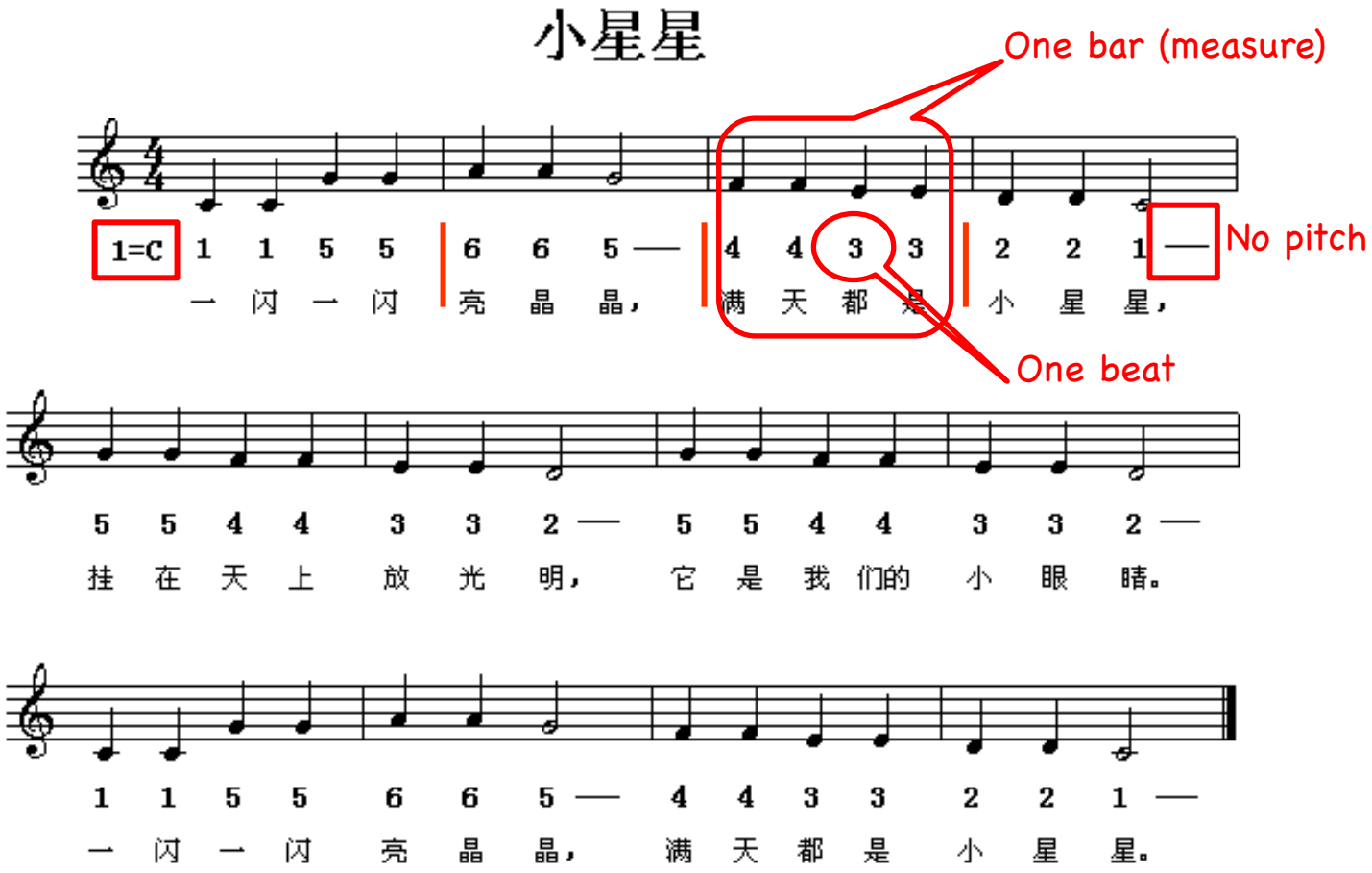
Agenda

- Announcements
- Keyboard Module: SampleDisplay
- Audio Tutorial
  - Concepts
  - Demo 1
  - Demo 2

Today's class will help you:

1. Understand how to revise the provided SampleDisplay module
2. Understand the concept of the Audio module
3. Understand how to revise the Audio module

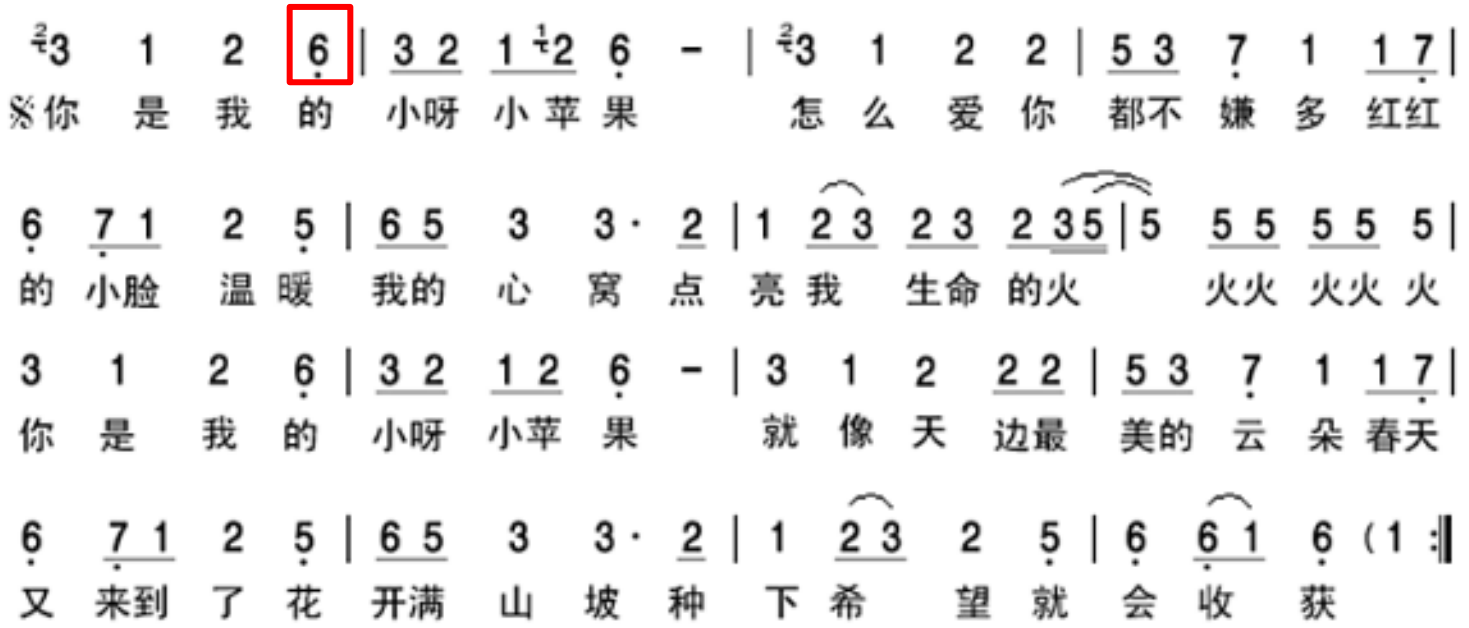
Numbered Musical Notation 1



Numbered Musical Notation 3

小蘋果

1 = ♭B 4/4 125 beats/min.

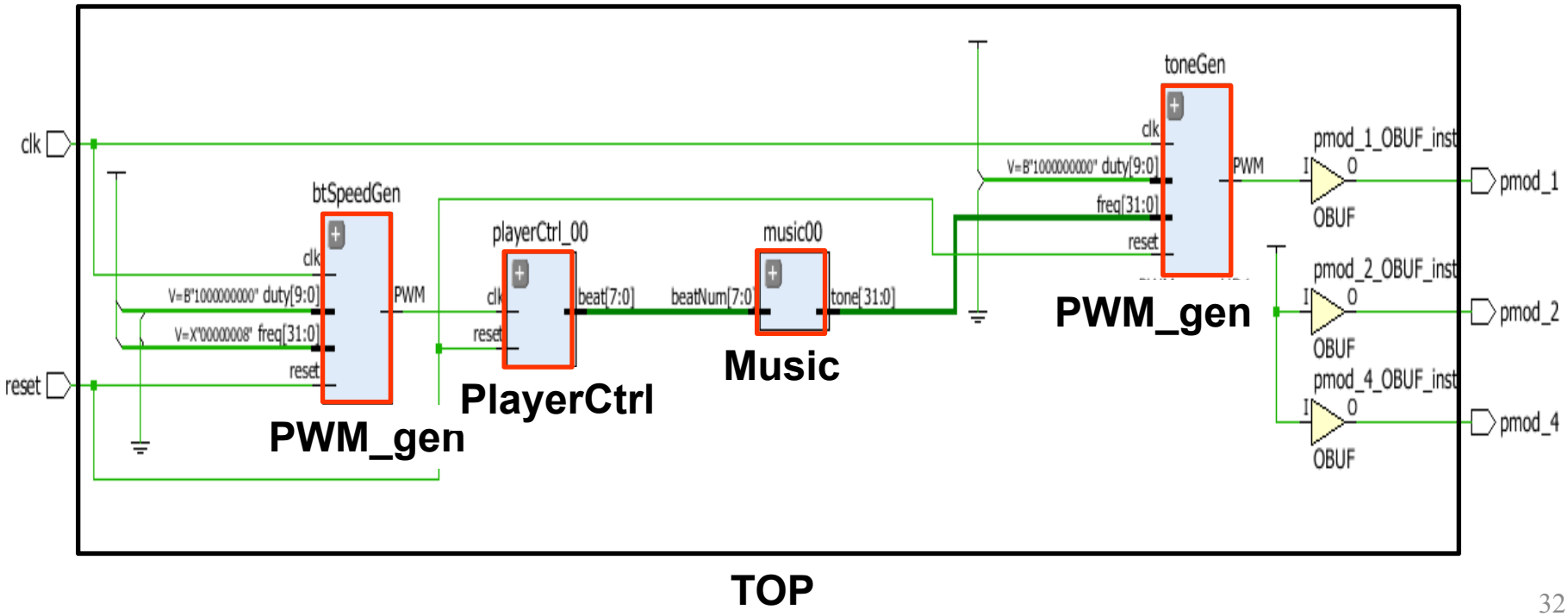


Demo 2: Music Box

- Repeatedly playing four bars (measures) of 小蘋果

Demo 2: Block Diagram

- In the demo code, only **pin 1**, **pin 2**, and **pin 4** are used
  - **Pin 1**: AIN
  - **Pin 2**: GAIN
  - **Pin 4**: SHUTDOWN_N



Demo 2: Top Module

```
module TOP (
    input clk,
    input reset,
    output pmod_1,
    output pmod_2,
    output pmod_4
);
parameter BEAT_FREQ = 32'd8;     //one beat=0.125sec
parameter DUTY_BEST = 10'd512;   //duty cycle=50%

wire [31:0] freq;
wire [7:0] ibeatNum;
wire beatFreq;

assign pmod_2 = 1'd1;    //no gain(6dB)
assign pmod_4 = 1'd1;    //turn-on
```

- Please note that this number represents the frequency
- 32'd8 -> 8Hz -> (period is 0.125 sec)

```
//Generate beat speed
PWM_gen btSpeedGen ( .clk(clk),
                     .reset(reset),
                     .freq(BEAT_FREQ),
                     .duty(DUTY_BEST),
                     .PWM(beatFreq)
);

//manipulate beat
PlayerCtrl playerCtrl_00 ( .clk(beatFreq),
                           .reset(reset),
                           .ibeat(ibeatNum)
);

//Generate variant freq. of tones
Music music00 ( .ibeatNum(ibeatNum),
                .tone(freq)
);

// Generate particular freq. signal
PWM_gen toneGen ( .clk(clk),
                  .reset(reset),
                  .freq(freq),
                  .duty(DUTY_BEST),
                  .PWM(pmod_1)
);
endmodule
```

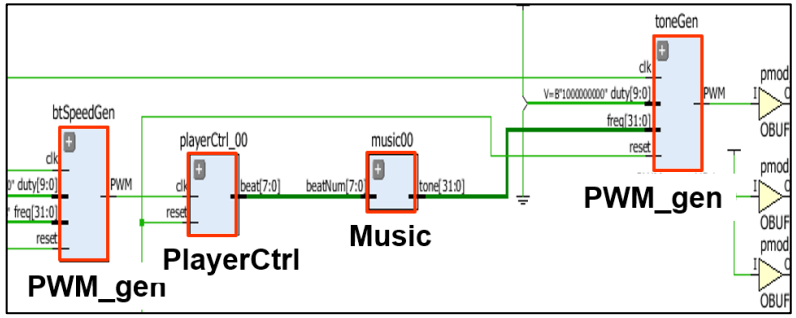
- Four modules are used in the design
- You can change this module and use it in your final project

Demo 2: BtSpeedGen Module

```
module PWM_gen (
    input wire clk,
    input wire reset,
    input [31:0] freq,
    input [9:0] duty,
    output reg PWM
);

wire [31:0] count_max = 100_000_000 / freq;
wire [31:0] count_duty = count_max * duty / 1024;
reg [31:0] count;

always @(posedge clk, posedge reset) begin
    if (reset) begin
        count <= 0;
        PWM <= 0;
    end else if (count < count_max) begin
        count <= count + 1;
        if(count < count_duty)
            PWM <= 1;
        else
            PWM <= 0;
    end else begin
        count <= 0;
        PWM <= 0;
    end
end

endmodule
```

Like clock divider !!

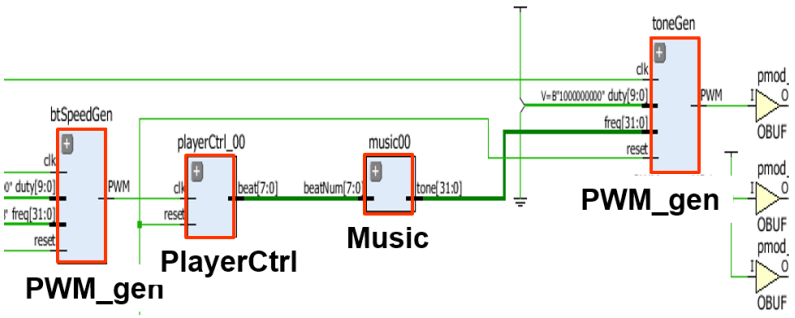
- Generates a clock to serve as the **tempo** of the music
- The module is the same as the **PWM_gen** module
- The first PWM_gen generates the **tempo**, while the second one generates the **tone**



Demo 2: PlayerCtrl Module

- Generate ibeat for the toneGen module
  - ibeat serves as an index for the **music module**
  - The input clk determines the **tempo** (**speed of beats**) of your music
  - The **tempo** is used to select the music tones in the **music module**

```
module PlayerCtrl (
    input clk,
    input reset,
    output reg [7:0] ibeat
);
parameter BEATLEAGTH = 63;

always @(posedge clk, posedge reset) begin
    if (reset)
        ibeat <= 0;
    else if (ibeat < BEATLEAGTH)
        ibeat <= ibeat + 1;
    else
        ibeat <= 0;
end

endmodule
```

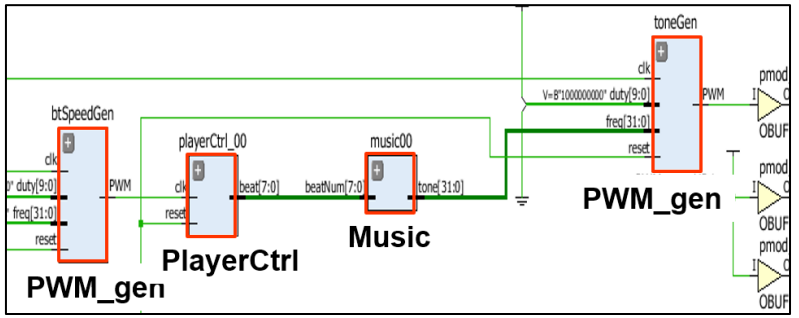


Demo 2: Music Module (1/2)

```
`define NM1 32'd466 //bB_freq
`define NM2 32'd523 //C_freq
`define NM3 32'd587 //D_freq
`define NM4 32'd622 //bE_freq
`define NM5 32'd698 //F_freq
`define NM6 32'd784 //G_freq
`define NM7 32'd880 //A_freq
`define NM0 32'd20000 //slience (over freq.)

module Music (
    input [7:0] ibeatNum,
    output reg [31:0] tone
);

always @(*) begin
    case (ibeatNum)      // 1/4 beat
        8'd0 : tone = `NM0; //3
        8'd1 : tone = `NM3;
        8'd2 : tone = `NM3;
        8'd3 : tone = `NM3;
        8'd4 : tone = `NM1; //1
        8'd5 : tone = `NM1;
        8'd6 : tone = `NM1;
        8'd7 : tone = `NM1;
        8'd8 : tone = `NM2; //2
        8'd9 : tone = `NM2;
        8'd10 : tone = `NM2;
        8'd11 : tone = `NM2;
        8'd12 : tone = `NM6 >> 1;    //6-
        8'd13 : tone = `NM6 >> 1;
        8'd14 : tone = `NM6 >> 1;
        8'd15 : tone = `NM6 >> 1;
```

- Similar to the previous "Decoder" module
- Generates tone frequencies for the toneGen module
- Use a **case statement** to select the tone for each beat
- You can change the ibeatNum to extend the length of your music

It could be useful in this "Lab"



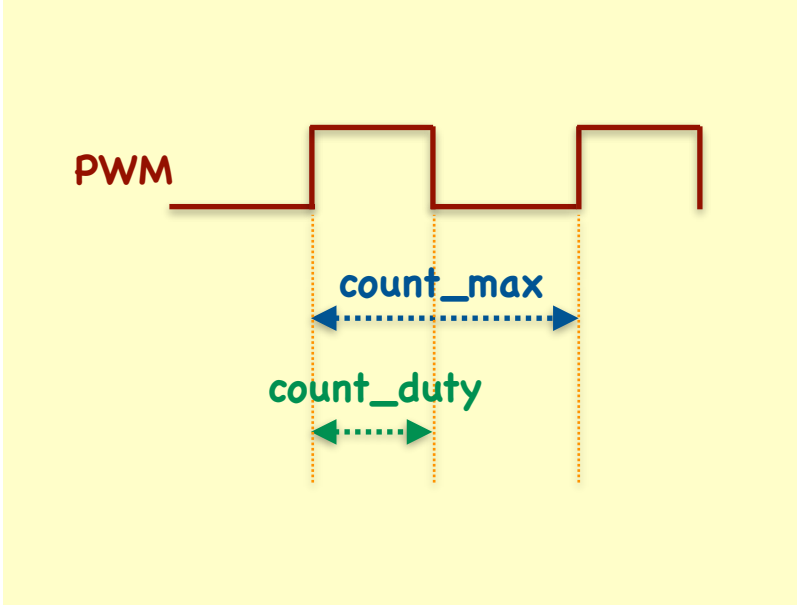
Demo 2: Music Module (2/2)

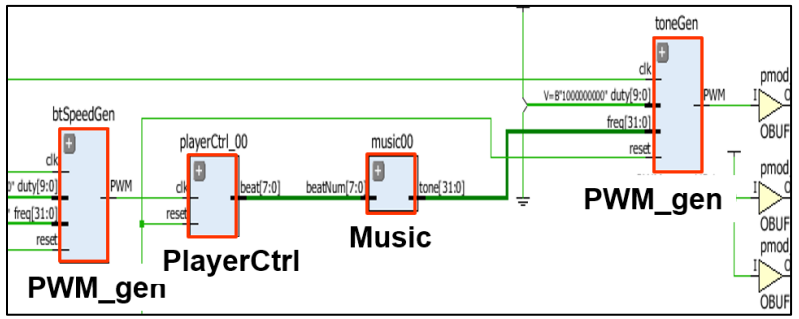
```
        8'd16 : tone = `NM3;
        8'd17 : tone = `NM3;
        8'd18 : tone = `NM2;
        8'd19 : tone = `NM2;
        8'd20 : tone = `NM1;
        8'd21 : tone = `NM1;
        8'd22 : tone = `NM2;
        8'd23 : tone = `NM2;
        8'd24 : tone = `NM6 >> 1;
        8'd25 : tone = `NM6 >> 1;
        8'd26 : tone = `NM6 >> 1;
        8'd27 : tone = `NM6 >> 1;
        8'd28 : tone = `NM0;
        8'd29 : tone = `NM0;
        8'd30 : tone = `NM0;
        8'd31 : tone = `NM0;

        8'd32 : tone = `NM3;
        8'd33 : tone = `NM3;
        8'd34 : tone = `NM3;
        8'd35 : tone = `NM3;
        8'd36 : tone = `NM1;
        8'd37 : tone = `NM1;
        8'd38 : tone = `NM1;
        8'd39 : tone = `NM1;
        8'd40 : tone = `NM2;
        8'd41 : tone = `NM2;
        8'd42 : tone = `NM2;
        8'd43 : tone = `NM2;
        8'd44 : tone = `NM2;
        8'd45 : tone = `NM2;
        8'd46 : tone = `NM2;
        8'd47 : tone = `NM2;
        8'd48 : tone = `NM5;
        8'd49 : tone = `NM5;
        8'd50 : tone = `NM3;
        8'd51 : tone = `NM3;
        8'd52 : tone = `NM7 >> 1;
        8'd53 : tone = `NM7 >> 1;
        8'd54 : tone = `NM7 >> 1;
        8'd55 : tone = `NM7 >> 1;
        8'd56 : tone = `NM1;
        8'd57 : tone = `NM1;
        8'd58 : tone = `NM1;
        8'd59 : tone = `NM1;
        8'd60 : tone = `NM1;
        8'd61 : tone = `NM1;
        8'd62 : tone = `NM7 >> 1;
        8'd63 : tone = `NM7 >> 1;
        default : tone = `NM0;
    endcase
end
```

- Note that you can definitely change the number of bits of **ibeatNum** to extend the length of your music

Demo 2: ToneGen Module

```
module PWM_gen (
    input wire clk,
    input wire reset,
    input [31:0] freq,
    input [9:0] duty,
    output reg PWM
);

wire [31:0] count_max = 100_000_000 / freq;
wire [31:0] count_duty = count_max * duty / 1024;
reg [31:0] count;

always @(posedge clk, posedge reset) begin
    if (reset) begin
        count <= 0;
        PWM <= 0;
    end else if (count < count_max) begin
        count <= count + 1;
        if(count < count_duty)
            PWM <= 1;
        else
            PWM <= 0;
    end else begin
        count <= 0;
        PWM <= 0;
    end
end

endmodule
```





Thank you for your attention!

Picture was taken by Chun-Yi Lee himself, who was an amateur photographer