Speakers

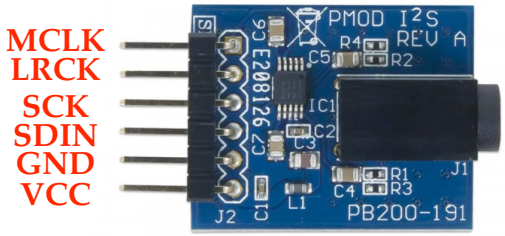
Hsi-Pin Ma

https://eeclass.nthu.edu.tw/course/18498
Department of Electrical Engineering
National Tsing Hua University

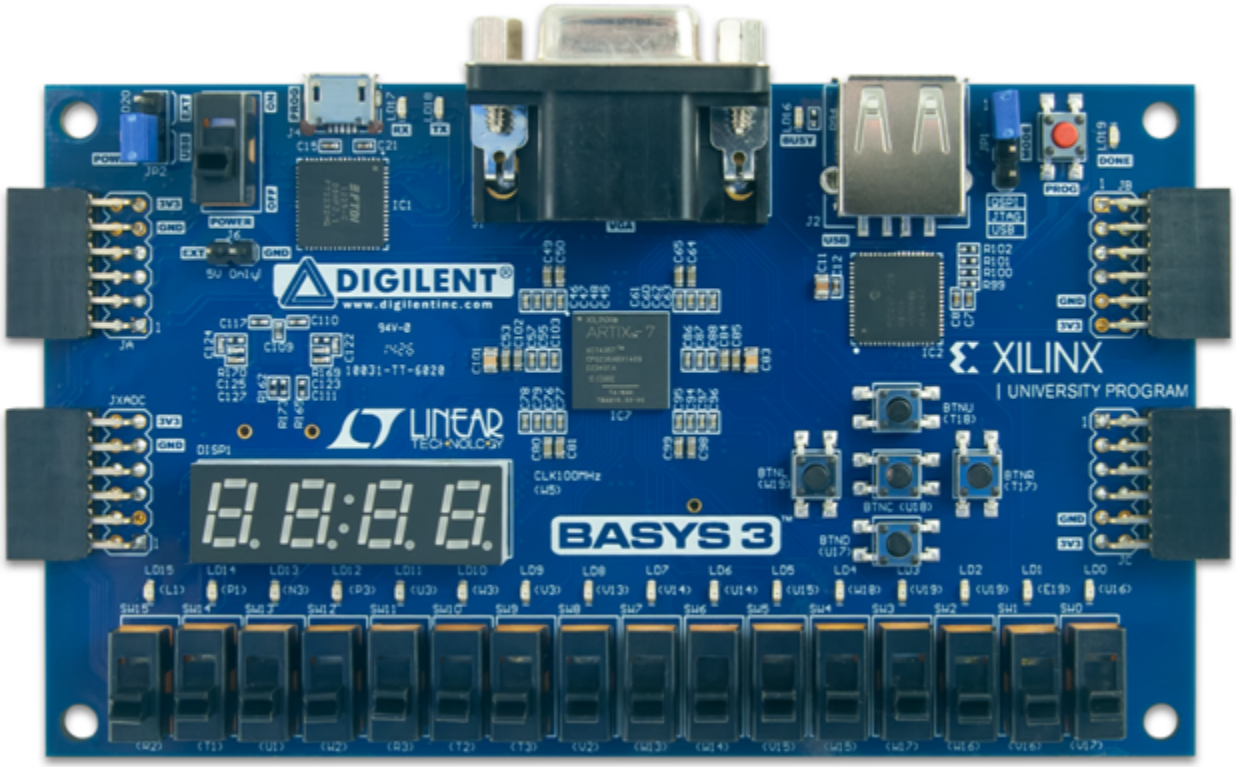
樂音三要素

- 響度 loudness -> amplitude
- 音調 pitch -> frequency
- 音色 timbre

Pmod I2S: Stereo Audio Output



Basys3: Pmod Pin-Out Diagram



JA12 : PWR
JA11 : GND
JA10: G3
JA9: H1
JA8 : K2
JA7: H1

JA6: PWR
JA5: GND
JA4: G2
JA3: J2
JA2: L2
JA1: J1

JXAC12 : PWR
JXAC11 : GND
JXAC10: N1
JXAC9: M1
JXAC8 : M3
JXAC7: K3

JXAC6: PWR
JXAC5: GND
JXAC4: N2
JXAC3: M2
JXAC2: L3
JXAC1: J3

upper row

JB1: A14
JB2: A16
JB3: B15
JB4: B16
JB5: GND
JB6: PWR

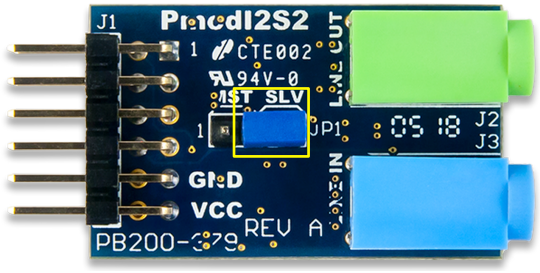
JB7: A15
JB8: A17
JB9: C15
JB10: C16
JB11: GND
JB12: PWR

lower row

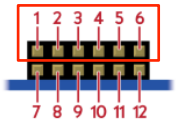
JC1: K17
JC2: M18
JC3: N17
JC4: P18
JC5: GND
JC6: PWR

JC7: L17
JC8: M19
JC9: P17
JC10: R18
JC11: GND
JC12: PWR

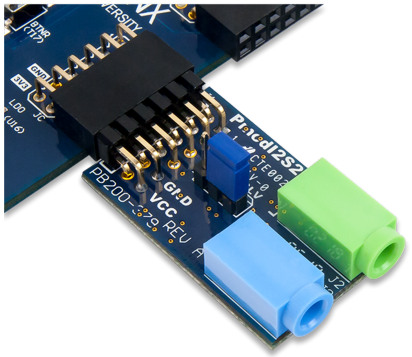
Pmod I2S2: Stereo Audio Input and Output

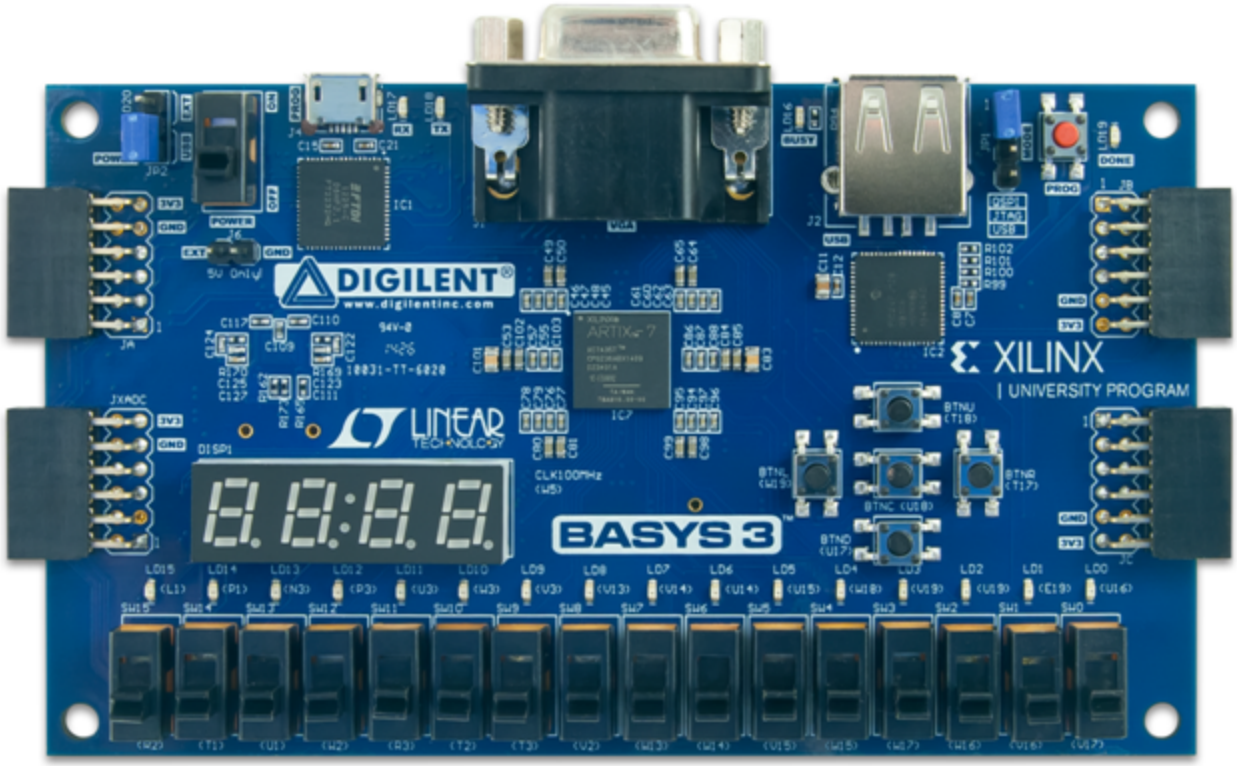


JP1: SLV mode



Pin 1	D/A MCLK
Pin 2	D/A LRCK
Pin 3	D/A SCLK
Pin 4	D/A SDIN
Pin 5	GND
Pin 6	VCC
Pin 7	A/D MCLK
Pin 8	A/D LRCK
Pin 9	A/D SCLK
Pin 10	A/D SDOUT
Pin 11	GND
Pin 12	VCC





JA12 : PWR
JA11 : GND
JA10: G3
JA9: H1
JA8 : K2
JA7: H1

JA6: PWR
JA5: GND
JA4: G2
JA3: J2
JA2: L2
JA1: J1

JXAC12 : PWR
JXAC11 : GND
JXAC10: N1
JXAC9: M1
JXAC8 : M3
JXAC7: K3

JXAC6: PWR
JXAC5: GND
JXAC4: N2
JXAC3: M2
JXAC2: L3
JXAC1: J3

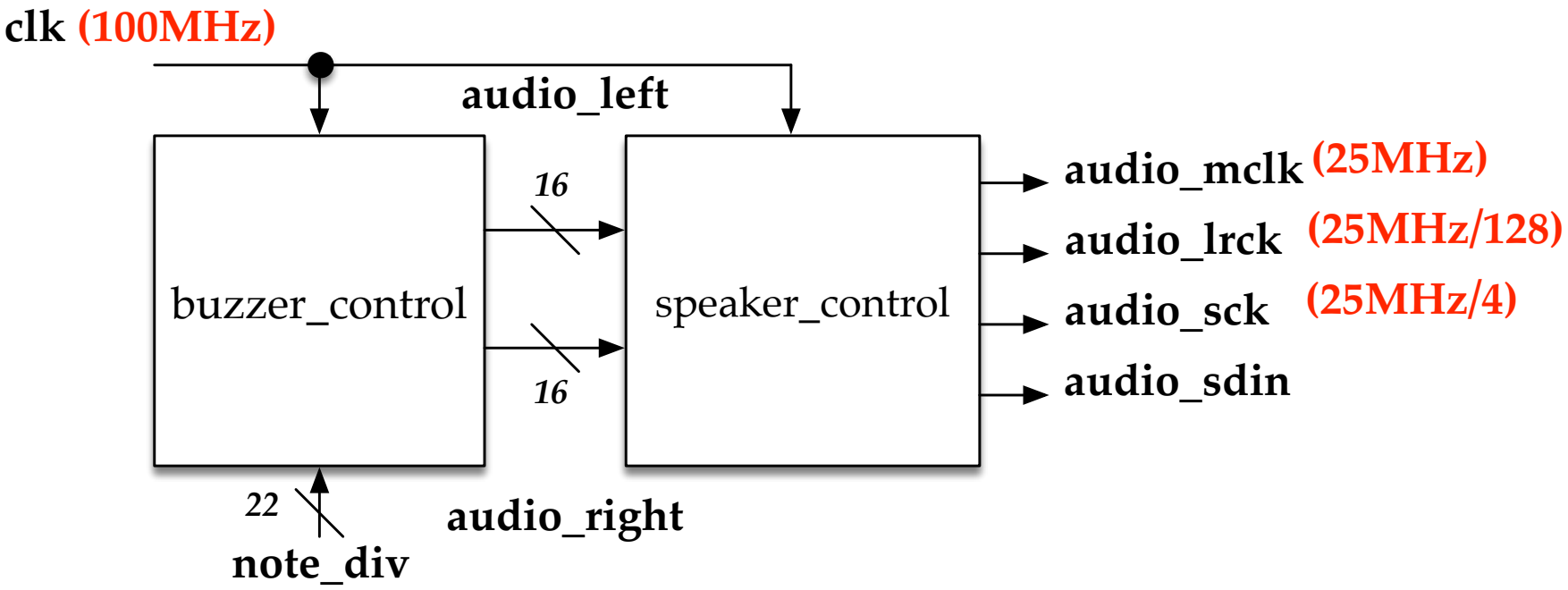
upper row

JB1: A14	JB7: A15
JB2: A16	JB8: A17
JB3: B15	JB9: C15
JB4: B16	JB10: C16
JB5: GND	JB11: GND
JB6: PWR	JB12: PWR

lower row

JC1: K17	JC7: L17
JC2: M18	JC8: M19
JC3: N17	JC9: P17
JC4: P18	JC10: R18
JC5: GND	JC11: GND
JC6: PWR	JC12: PWR

Speaker


clk (100MHz)
audio_left
buzzer_control
16
16
speaker_control
audio_mclk (25MHz)
audio_lrck (25MHz/128)
audio_sck (25MHz/4)
audio_sdin
22
note_div
audio_right

Buzzer Control

- The buzzer frequency is obtained by dividing crystal frequency 100MHz by N.
- The buzzer clock (b_clk) is periodically inverted for every N/2 clock cycles. (***determine the pitch***)
- Note frequency
  - Mid Do: 261 Hz
  - Mid Re: 293 Hz
  - Mid Mi: 330 Hz

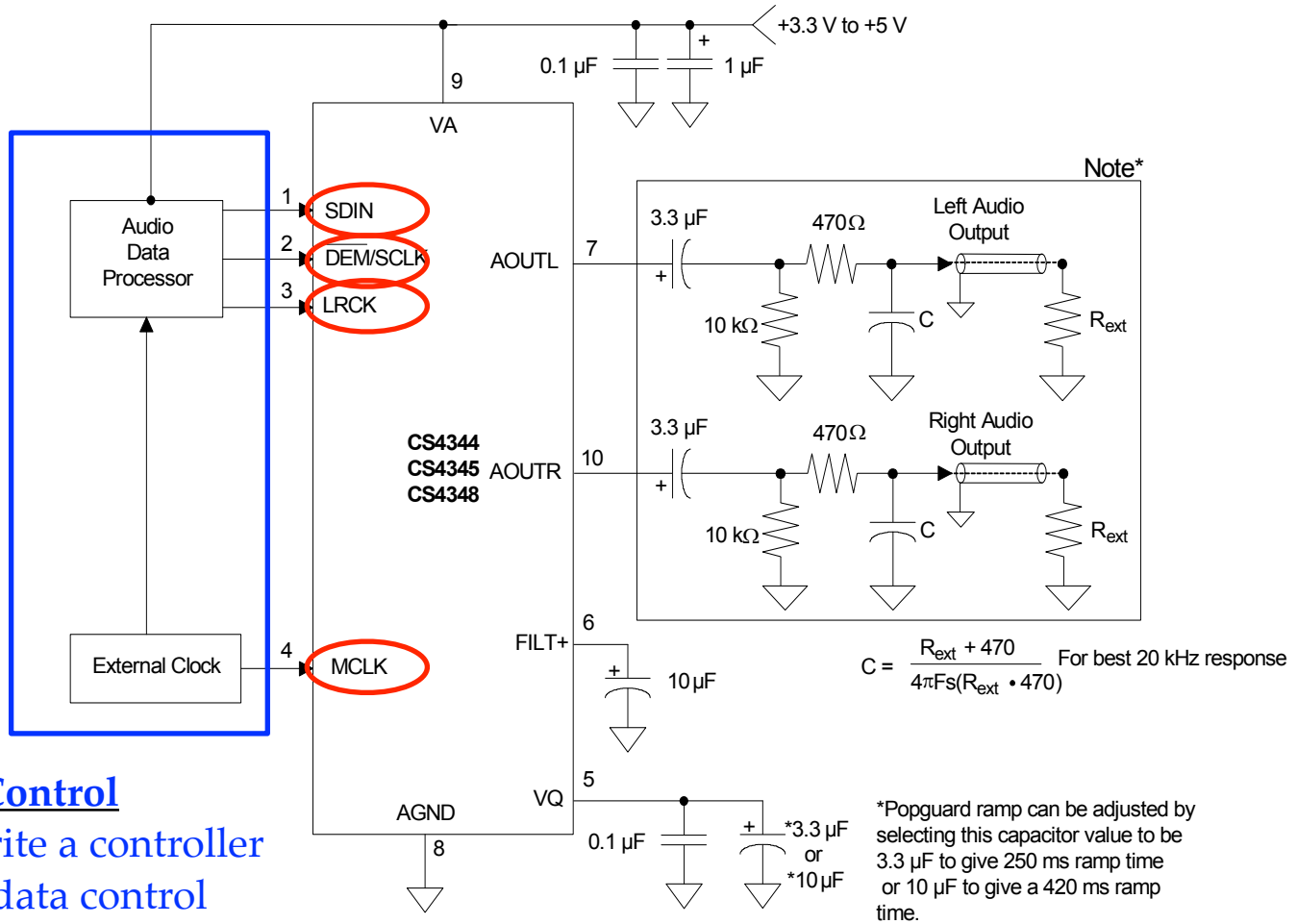
Buzzer Control

```
module note_gen(
  clk, // clock from crystal
  rst_n, // active low reset
  note_div, // div for note generation
  audio_left, // left sound audio
  audio_right // right sound audio
);

// I/O declaration
input clk; // clock from crystal
input rst_n; // active low reset
input [21:0] note_div; // div for note generation
output [15:0] audio_left; // left sound audio
output [15:0] audio_right; // right sound audio

// Declare internal signals
reg [21:0] clk_cnt_next, clk_cnt;
reg b_clk, b_clk_next;
```

```
// Note frequency generation
always @(posedge clk or negedge rst_n)
  if (~rst_n)
  begin
    clk_cnt <= 22'd0;
    b_clk <= 1'b0;
  end
  else
  begin
    clk_cnt <= clk_cnt_next;
    b_clk <= b_clk_next;
  end
always @*
  if (clk_cnt == note_div)
  begin
    clk_cnt_next = 22'd0;
    b_clk_next = ~b_clk;
  end
  else
  begin
    clk_cnt_next = clk_cnt + 1'b1;
    b_clk_next = b_clk;
  end

// Assign the amplitude of the note
assign audio_left = (b_clk == 1'b0) ? 16'hB000 : 16'h5FFF;
assign audio_right = (b_clk == 1'b0) ? 16'hB000 : 16'h5FFF;


endmodule
```

Speaker Control

- Control the DAC (digital to analog converter) CS4344



*Popguard ramp can be adjusted by selecting this capacitor value to be 3.3 µF to give 250 ms ramp time or 10 µF to give a 420 ms ramp time.

Speaker Control

Should write a controller for audio data control (output SDIN, LRCK, MCLK, SCK)

Speaker Control

- Control the DAC (digital to analog converter) CS4344
  - Internal SCK mode: 16-bit data and SCK = 32*Fs if MCLK/LRCK = 1024, 512, 256, 128, or 64
    - MCLK (Master Clock) synchronizes the audio data transmission **25MHz (~24.5760MHz)**
    - MCLK/LRCK must be an integer ratio **128**
    - LRCK (Left-Right Clock, or Word Select (WS) Clock, or Sample Rate (Fs) Clock) controls the sequence (left or right) of the serial stereo output **25MHz/128 (~192kHz)**
    - Serial Clock (SCK) controls the shifting of data into the input data buffers (32*Fs) **25MHz/128*32 = 25MHz/4**

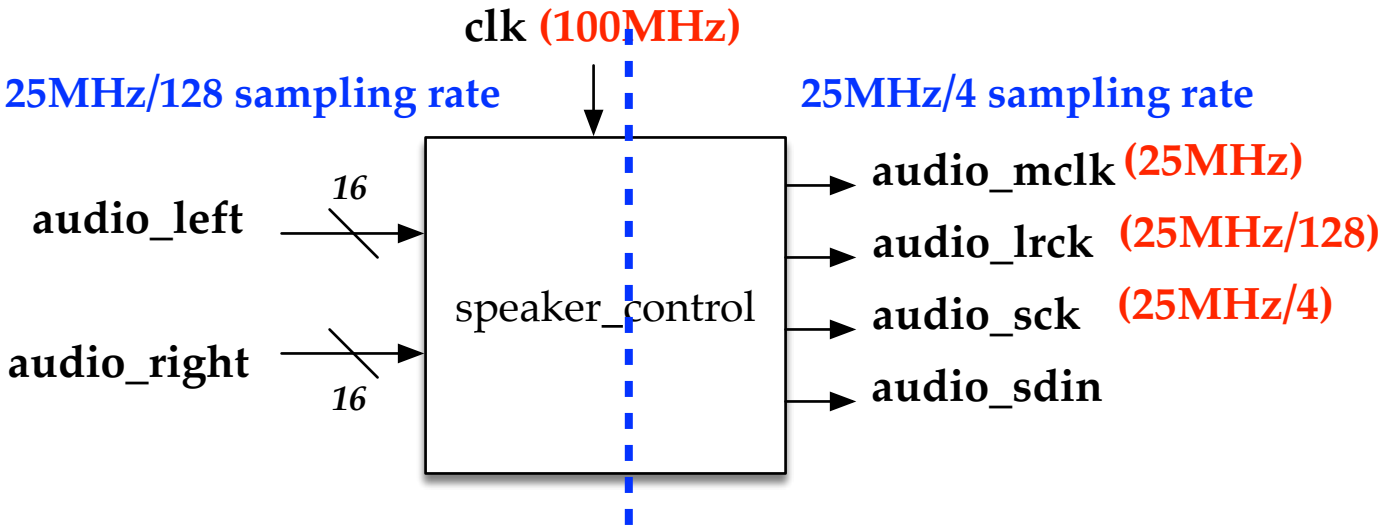
LRCK (kHz)	MCLK (MHz)									
	64x	96x	128x	192x	256x	384x	512x	768x	1024x	1152x
32	-	-	-	-	8.1920	12.2880	-	-	32.7680	36.8640
44.1	-	-	-	-	11.2896	16.9344	22.5792	33.8680	45.1580	-
48	-	-	-	-	12.2880	18.4320	24.5760	36.8640	49.1520	-
64	-	-	8.1920	12.2880	-	-	32.7680	49.1520	-	-
88.2	-	-	11.2896	16.9344	22.5792	33.8680	-	-	-	-
96	-	-	12.2880	18.4320	24.5760	36.8640	-	-	-	-
128	8.1920	12.2880	-	-	32.7680	49.1520	-	-	-	-
176.4	11.2896	16.9344	22.5792	33.8680	-	-	-	-	-	-
192	12.2880	18.4320	24.5760	36.8640	-	-	-	-	-	-
Mode	QSM				DSM		SSM			

Speaker Control

- Input (stereo audio ***parallel input***)
  - audio_left [15:0] / audio_right[15:0]
  - 16'h8000(min) ~ 16'h7FFF (max) (2's complement)

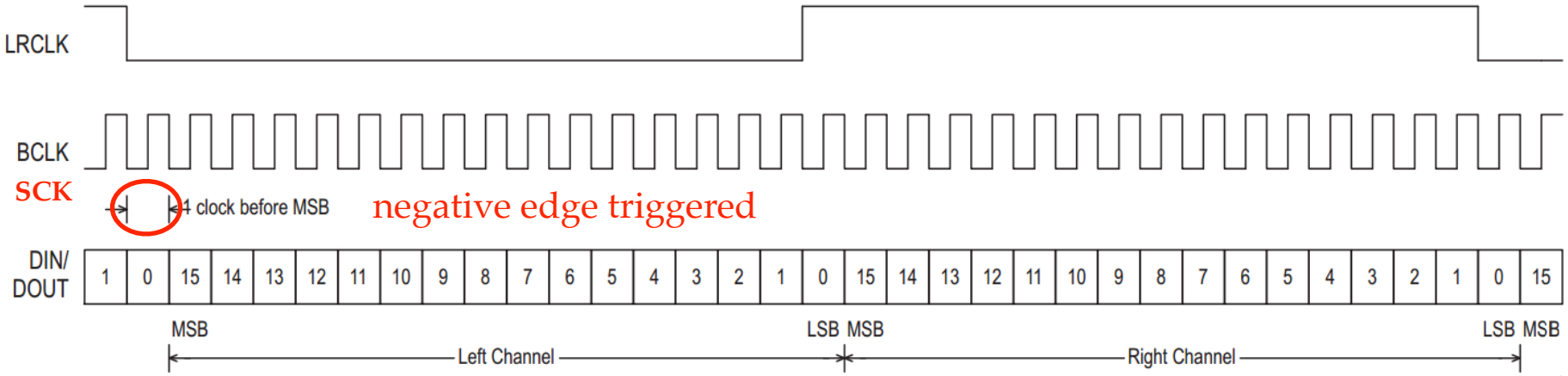
Operation Frequency different
25MHz/128 vs. 25MHz/4
Data rate the same: 6.25Mbps

- Output (stereo audio ***serial output***)
  - audio_mclk = 25MHz (divided by 4 from external crystal 100MHz)
  - audio_lrck = 25MHz / 128 (sample rate clock of parallel input audio)
  - audio_sck = 25MHz / 4 (serial clock)
  - audio_sdin (1 bit serial audio data output)



Speaker Control (I2S Protocol)

- Frequency dividers
  - audio_mclk
  - audio_lrck
  - audio_sck
- Parallel to serial module
  - To re-formulate the audio sequence
    - Left first, then right
    - MSB first
    - one SCK cycle delayed



speaker.v

```
module speaker(
  clk, // clock from crystal
  rst_n, // active low reset
  audio_mclk, // master clock
  audio_lrck, // left-right clock
  audio_sck, // serial clock
  audio_sdin // serial audio data input
);

// I/O declaration
input clk;  // clock from the crystal
input rst_n;  // active low reset
output audio_mclk; // master clock
output audio_lrck; // left-right clock
output audio_sck; // serial clock
output audio_sdin; // serial audio data input
// Declare internal nodes
wire [15:0] audio_in_left, audio_in_right;

// Note generation
buzzer_control Ung(
  .clk(clk), // clock from crystal
  .rst_n(rst_n), // active low reset
  .note_div(22'd191571), // div for note generation
  .audio_left(audio_in_left), // left sound audio
  .audio_right(audio_in_right) // right sound audio
);
```

speaker.v

```
// Speaker controllor
speaker_control Usc(
  .clk(clk),  // clock from the crystal
  .rst_n(rst_n),  // active low reset
  .audio_in_left(audio_in_left), // left channel audio data input
  .audio_in_right(audio_in_right), // right channel audio data input
  .audio_mclk(audio_mclk), // master clock
  .audio_lrck(audio_lrck), // left-right clock
  .audio_sck(audio_sck), // serial clock
  .audio_sdin(audio_sdin) // serial audio data input
);

endmodule
```

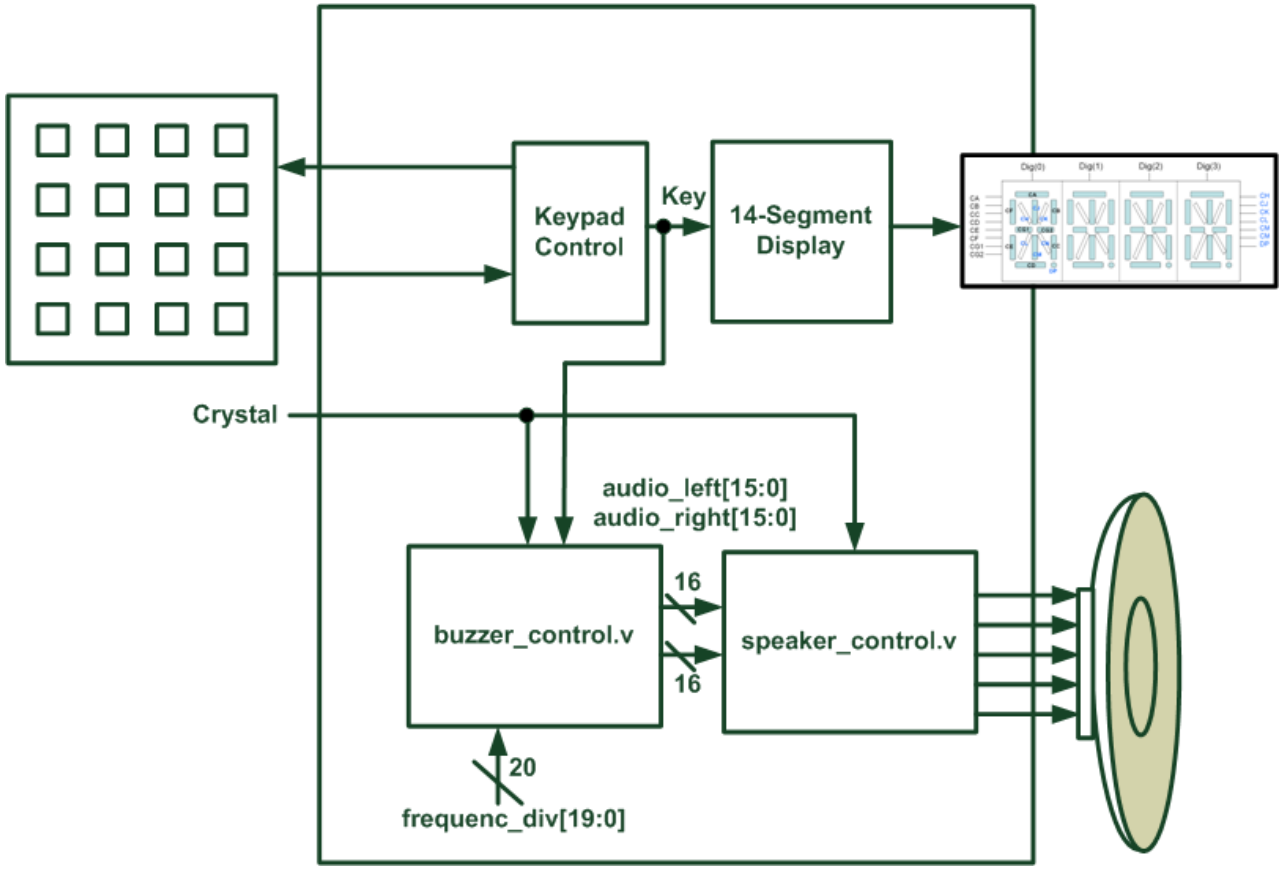
speaker.xdc

```
# Clock
set_property PACKAGE_PIN W5 [get_ports {clk}]
set_property IOSTANDARD LVCMOS33 [get_ports {clk}]

# active low reset
set_property PACKAGE_PIN V17 [get_ports {rst_n}]
set_property IOSTANDARD LVCMOS33 [get_ports {rst_n}]

# Pmod I2S      Use upper row of JB
set_property PACKAGE_PIN A14 [get_ports {audio_mclk}]
set_property IOSTANDARD LVCMOS33 [get_ports {audio_mclk}]
set_property PACKAGE_PIN A16 [get_ports {audio_lrck}]
set_property IOSTANDARD LVCMOS33 [get_ports {audio_lrck}]
set_property PACKAGE_PIN B15 [get_ports {audio_sck}]
set_property IOSTANDARD LVCMOS33 [get_ports {audio_sck}]
set_property PACKAGE_PIN B16 [get_ports {audio_sdin}]
set_property IOSTANDARD LVCMOS33 [get_ports {audio_sdin}]
```

Note Frequency Table

Tone	Do	Re	Me	Fa	So	La	Si
Low (Hz)						220	245
Mid (Hz)	261	293	330	349	392	440	494
High (Hz)	524	588	660	698	784	880	988

Electronic Organ



Buzzer Control

```
module buzzer_control(
  clk, // clock from crystal
  rst_n, // active low reset
  note_div, // div for note generation
  audio_left, // left sound audio
  audio_right // right sound audio
);

// I/O declaration
input clk; // clock from crystal
input rst_n; // active low reset
input [19:0] note_div; // div for note generation
output [15:0] audio_left; // left sound audio
output [15:0] audio_right; // right sound audio

// Declare internal signals
reg [19:0] clk_cnt_next, clk_cnt;
reg b_clk, b_clk_next;
```

```
// Note frequency generation
always @(posedge clk or negedge rst_n)
  if (~rst_n)
  begin
    clk_cnt <= 20'd0;
    b_clk <= 1'b0;
  end
  else
  begin
    clk_cnt <= clk_cnt_next;
    b_clk <= b_clk_next;
  end
always @*
  if (clk_cnt == note_div)
  begin
    clk_cnt_next = 20'd0;
    b_clk_next = ~b_clk;
  end
  else
  begin
    clk_cnt_next = clk_cnt + 1'b1;
    b_clk_next = b_clk;
  end

// Assign the amplitude of the note
assign audio_left = (b_clk == 1'b0) ? 16'h4000 : 16'h3FFF;
assign audio_right = (b_clk == 1'b0) ? 16'h4000 : 16'h3FFF;


endmodule
```