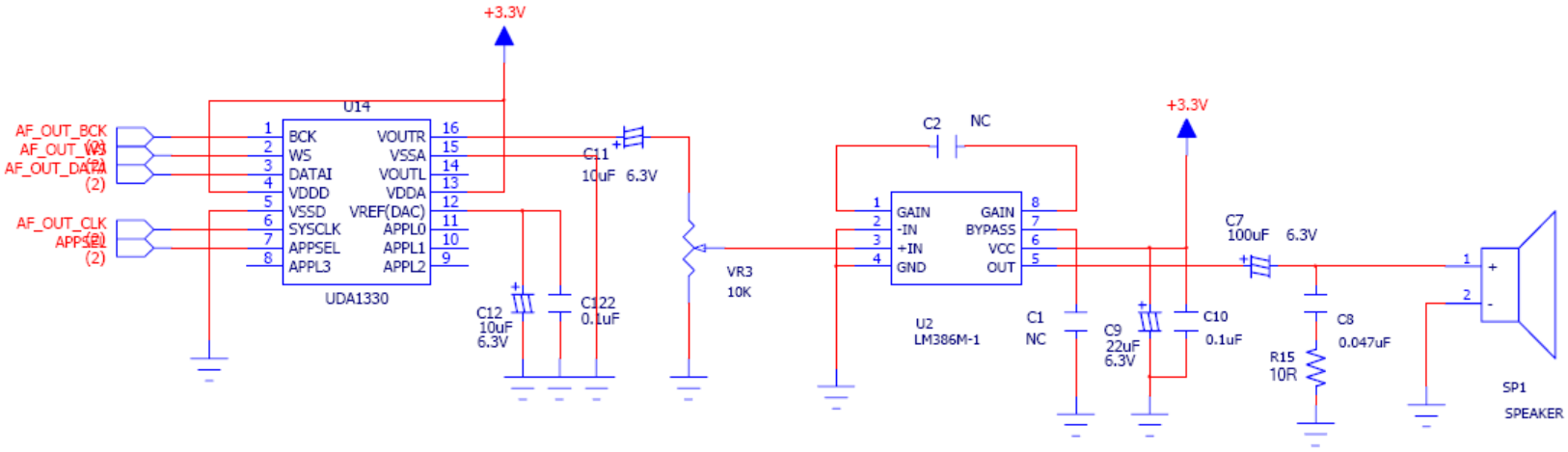
# Speaker

Hsi-Pin Ma

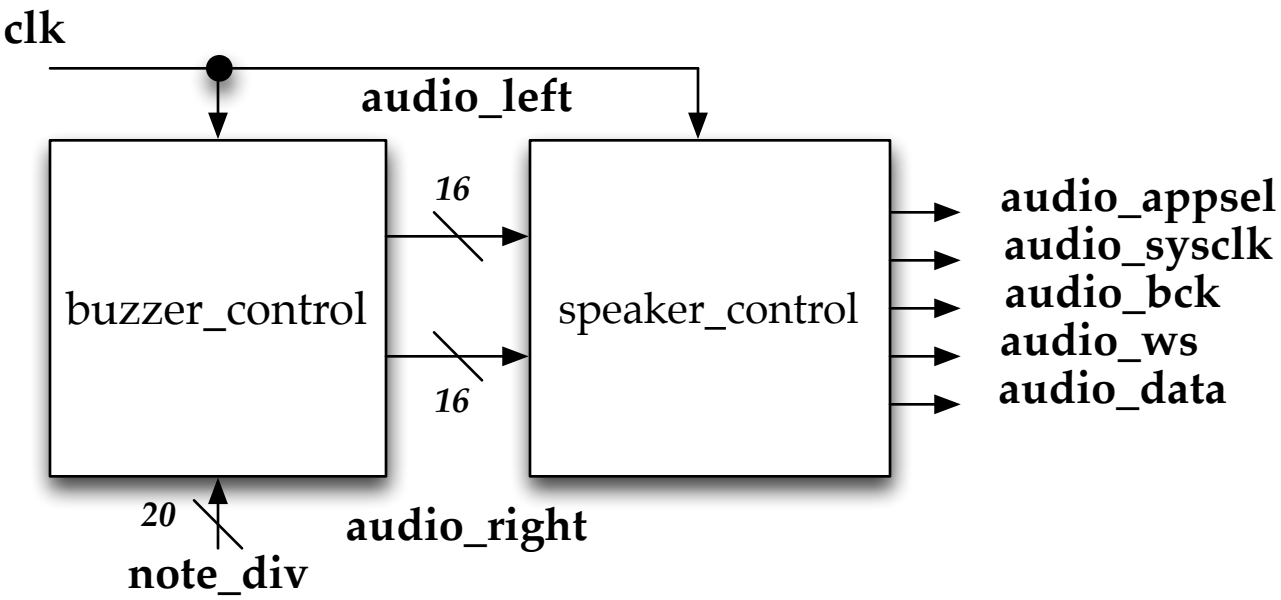
http://lms.nthu.edu.tw/course/21094
Department of Electrical Engineering
National Tsing Hua University

# Speaker

- Control the DAC (digital to analog converter)
  - Use WS (word select) to control the sequence (left or right) of the serial stereo output
  - Use BCK (bit clock) to synchronize the audio data transmission

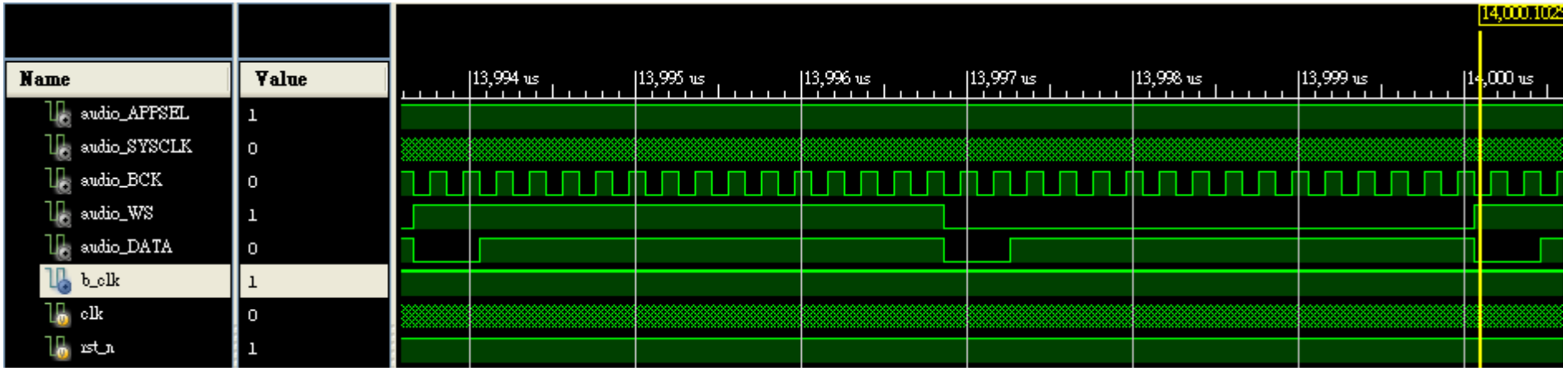


# Speaker


clk
audio_left
buzzer_control
16
16
speaker_control
audio_appsel
audio_sysclk
audio_bck
audio_ws
audio_data
20
note_div
audio_right

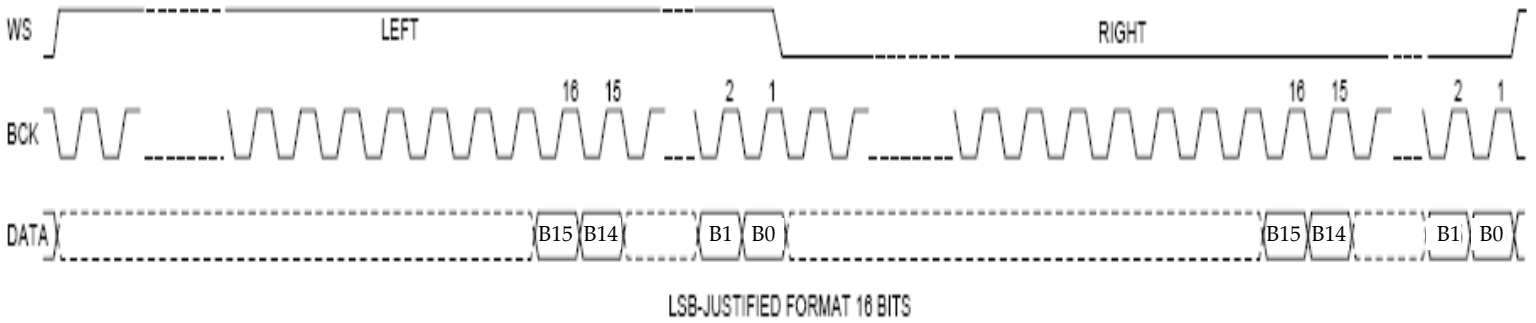
# Speaker Control

- Input (stereo audio *parallel input*)
  - audio_left [15:0] / audio_right[15:0]
- Output (stereo audio *serial output*)
  - audio_appsel = 1 (stereo playing)
  - audio_sysclk = 40MHz (from external crystal LOC = R10)
  - audio_bck = 5MHz (bit clock of audio data)
  - audio_ws = 1/0 (left/right, (5/32) MHz sampling rate)
  - audio_data = 16'h8000(min) ~ 16'h7FFF (max) (2's complement)



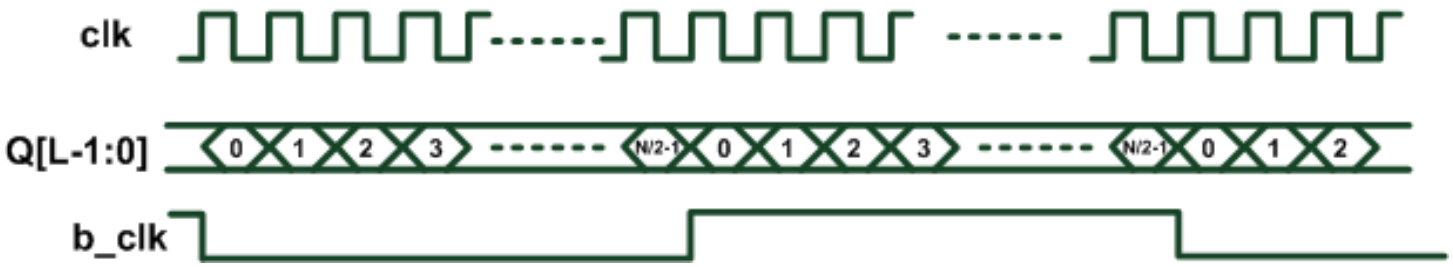
# Speaker Control

- Frequency dividers
  - audio_bck
  - audio_ws
- Parallel to serial module
  - To re-formulate the audio sequence
    - Right first, then left
    - MSB first



# Buzzer Control

- The buzzer frequency is obtained by dividing crystal frequency 40MHz by N.
- The buzzer clock (b_clk) is periodically inverted for every N/2 clock cycles. (***determine the sound***)
- Note frequency
  - Mid Do: 261 Hz
  - Mid Re: 293 Hz
  - Mid Mi: 330 Hz



# 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
```

# speaker.v

```
module speaker(
  clk, // clock from crystal
  rst_n, // active low reset
  audio_appsel, // playing mode selection
  audio_sysclk, // control clock for DAC (from crystal)
  audio_bck, // bit clock of audio data (5MHz)
  audio_ws, // left/right parallel to serial control
  audio_data // serial output audio data
);

// I/O declaration
input clk;  // clock from the crystal
input rst_n;  // active low reset
output audio_appsel; // playing mode selection
output audio_sysclk; // control clock for DAC (from crystal)
output audio_bck; // bit clock of audio data (5MHz)
output audio_ws; // left/right parallel to serial control
output audio_data; // serial output audio data
// Declare internal nodes
wire [15:0] audio_in_left, audio_in_right;

// Note generation
note_gen Ung(
  .clk(clk), // clock from crystal
  .rst_n(rst_n), // active low reset
  .note_div(20'd76628), // 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_appsel(audio_appsel), // playing mode selection
  .audio_sysclk(audio_sysclk), // control clock for DAC (from crystal)
  .audio_bck(audio_bck), // bit clock of audio data (5MHz)
  .audio_ws(audio_ws), // left/right parallel to serial control
  .audio_data(audio_data) // serial output audio data
);

endmodule
```

# speaker.ucf

```
NET   "audio_APPSEL"      LOC = H18;
NET   "audio_BCK“          LOC = K16;
NET   "audio_SYSCLK“       LOC = H17;
NET   "audio_WS"          LOC = L15;
NET   "audio_DATA"         LOC = L16;
NET   "clk"               LOC = R10;
NET   "rst_n“              LOC = T2;
```