情報工学実験 II
第 4 回： 電子ピアノの実装

柴田裕一郎・松尾堅太郎・元島晃伸

shibata@cis.nagasaki-u.ac.jp
情報工学コース

2019 年 1 月 11 日

電子ピアノの実装

- ♪ 音を出す仕組み
- ♪ 音の周波数と音階
- ♪ 組み合わせ回路と関数テーブル
- ♪ 自作電子ピアノを弾いてみよう

音を出すには

- 実験ボードには DAC（Digital Analog Converter）が搭載
  - 入力： 8 ビットのディジタルデータ（0〜255）
  - 出力： アナログの電圧値
- FPGA で時間とともに周期的に変化する 8 ビットのデータを出力
  - ディジタル値の波 → 「DAC」 → 電圧の波
  - 電圧の波 → 「スピーカー」 → 空気の波（音）

とりあえず音を出す回路（sound.sv）

- スイッチ入力（sw[7:0]）
  - 8 個のスイッチボタン
  - 押すと 1，離すと 0
  - 一番左が sw[0] で一番右が sw[7]
  - メタステーブルやチャタリングには対応済
- クロック入力（clk）
  - 周波数 1.5 MHz
- カウンタ（count[11:0]）
  - ボタンが押されている間は 1 クロックに 1 ずつカウントアップ
  - 4,095 まで行くとオーバフローして 0 に戻る
- データ出力（dout[7:0]）
  - カウンタの MSB が 1 なら 255，そうでないなら 0 を出力

sound モジュールの記述

```
1  `default_nettype none
2  module sound
3    (
4     input wire         clk,
5     input wire         rst,
6     input wire [7:0]   sw,
7     output wire [7:0] dout
8     );
9
10    reg [11:0]          count;
11
12    always_ff @(posedge clk) begin
13      if (rst)
14        count <= 12'd0;
15      else if (sw[0])
16        count <= count + 12'd1;
17    end
18
19    assign dout = count[11]? 8'd255: 8'd0;
20 endmodule
21 `default_nettype wire
```

シミュレーションスクリプト (1/2)

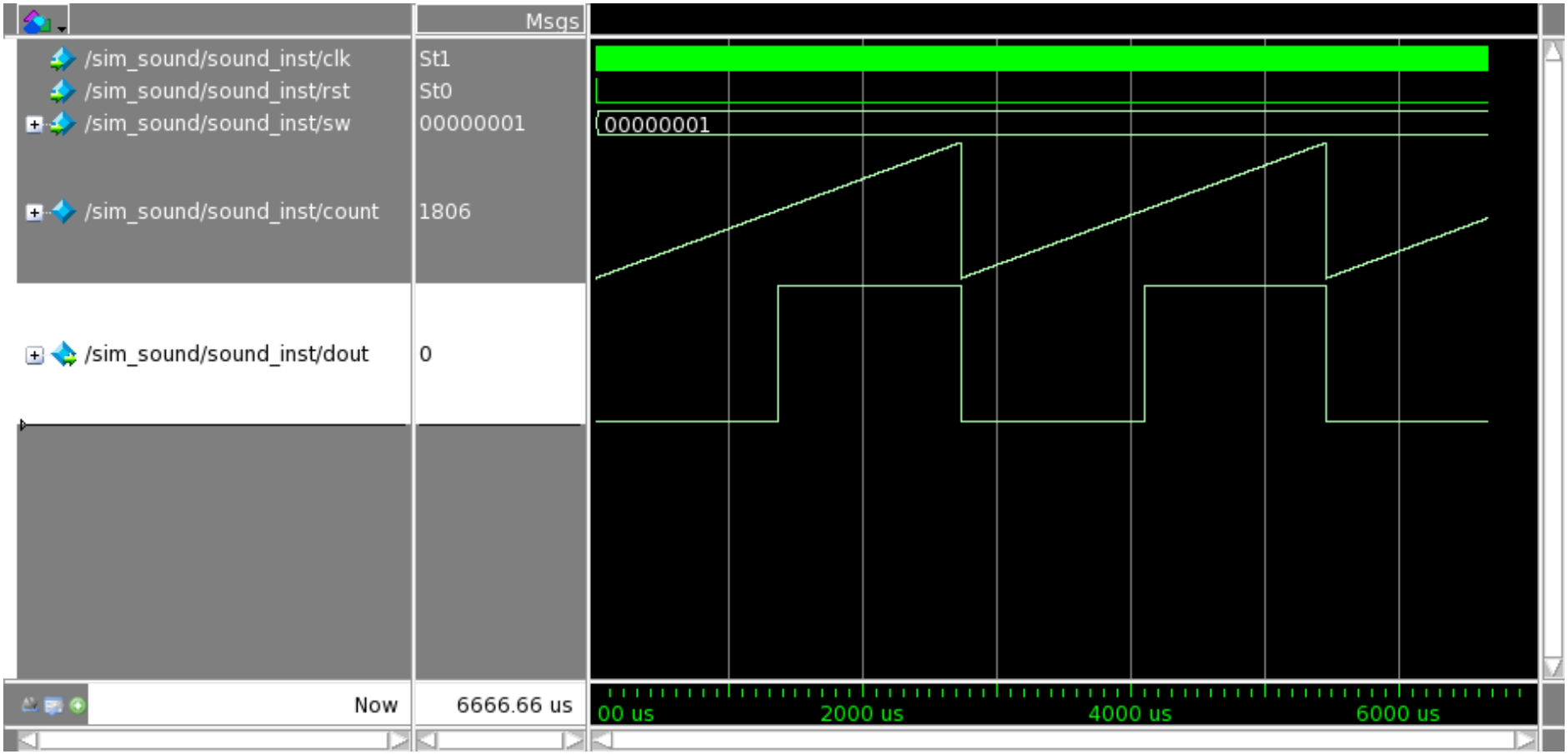
```
2  `timescale 1ns/1ps
3  module sim_sound();
4      localparam real    CLOCK_FREQ_HZ     = 1.5 * 10**6; // 1.5 MHz
5      localparam real    CLOCK_PERIOD_NS   = 10**9 / CLOCK_FREQ_HZ;
6      localparam integer SIMULATION_CYCLES = 10000;
7      logic              clk, rst;
8      logic [7:0]        sw;
9      wire [7:0]         dout;
10
11     sound sound_inst(.clk(clk), .rst(rst), .sw(sw), .dout(dout));
12
13     initial begin
14         clk <= 1'b0;
15         repeat (SIMULATION_CYCLES) begin
16          #(CLOCK_PERIOD_NS / 2.0)
17            clk <= 1'b1;
18          #(CLOCK_PERIOD_NS / 2.0)
19            clk <= 1'b0;
20            print();
21         end
22         $finish;
23     end
```

シミュレーションスクリプト (2/2)

```
25    initial begin
26       rst <= 1'b1;
27     #(CLOCK_PERIOD_NS)
28       rst <= 1'b0;
29    end
30
31    initial begin
32       sw <= 8'b00000001;
33    end
34
35    task print();
36       $write("time= %5d sw= %b count= %d dout= %d\n",
37              $time, sw, sound_inst.count, dout);
38    endtask
39 endmodule
40 `default_nettype wire
```

- ボタンを押した状態で 10,000 クロックシミュレーション

シミュレーション結果



- Radix を「Unsigned」に Format を「Analog (automatic)」に設定
- 周期 4,096 クロックの矩形波

定数パラメータ

- localparam で定数パラメータを定義
- real（実数型）や integer（整数型）で定義可能
- 利用例：

```
localparam real CLOCK_FREQ_HZ = 1.5 * 10**6;
```

以降「CLOCK_FREQ_HZ」と記述すると，実数型で 1.5×10^6 と記述したのと同じになる

練習

1： この音の周波数を計算せよ．ちなみに，個人差もかなりあるが，人間の可聴周波数はだいたい 20 Hz から 20 kHz 程度である．

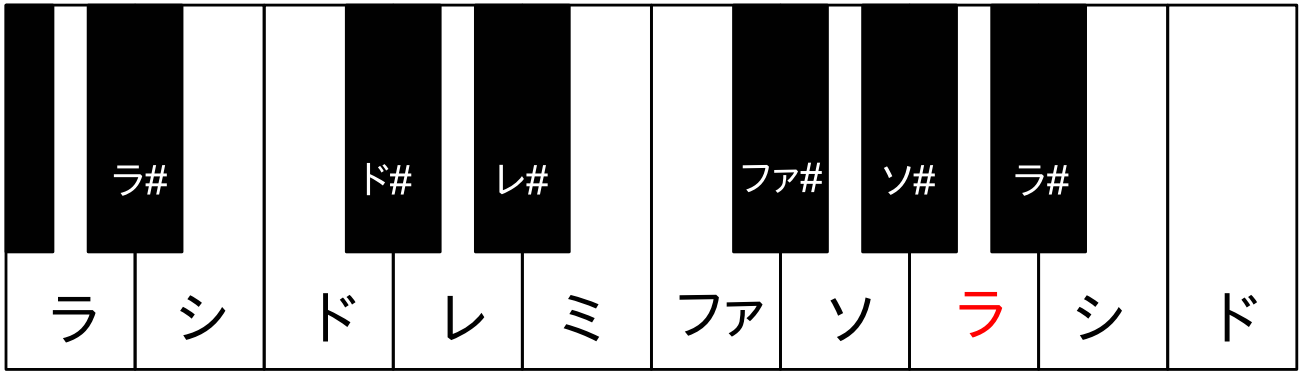
2： sound モジュールの SystemVerilog 記述の 16 行目を次のように変更し，1 クロックに 2 ずつ増やすように変更した．出力される音の周波数はいくらになるか?

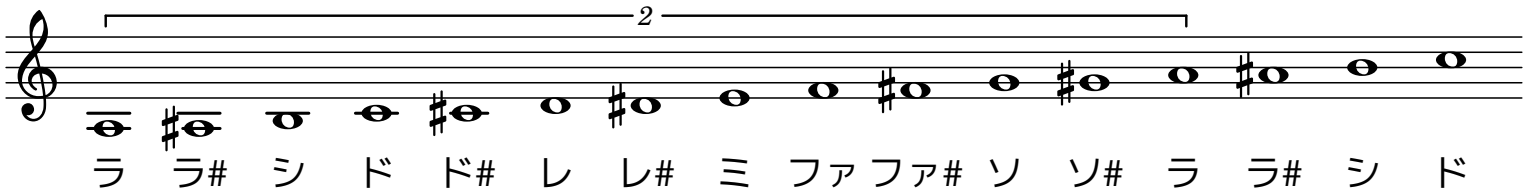
```
count <= count + 12'd2;
```

音階の仕組み

- 基準音は「ラ」
  - 国際標準では周波数 440 Hz
  - 楽器の場合は少し高めにチューニングすることが多い
- オクターブ
  - 周波数が 2 倍となる音の関係
  - 高い「ド」と低い「ド」の関係
- 平均律
  - ピアノの調律に用いられる音律
  - オクターブを等比に 12 分割

音階とピアノの鍵盤





練習

3： 基準の「ラ」音を 442 Hz とした平均律で調律されたピアノがある．白鍵の各音（ドレミファソラシド）の周波数を求めよ．

電子ピアノの仕様

- モジュール名は `piano`
- 入出力は sound モジュールと同じ
- クロックの周波数は 1.5 MHz
- 基準音（「ラ」）の周波数は 442 Hz
- 8 つの押しボタンスイッチを左から順にピアノの白鍵の各音に対応させる
- 出力波形は 8 ビットの正弦波（10 進数で 0 から 255 までの値をとる）
- スイッチを押している間は音が鳴り続け，スイッチを離すと音は止む
- 同時に複数のスイッチが押された場合は，高い方の音のスイッチを優先する
- 24 ビットのカウンタを使って音の高さをコントロールする

設計方針

- 24 ビットのカウンタを用意（counter[23:0]）
  - sound モジュールよりも低い周波数の波を作れる
  - スイッチによって増分を変化させる

練習： クロック周波数を 1.5 MHz とする．24 ビットのカウンタを 1 クロックに 1 ずつカウントアップした場合，出力音の周波数はいくらか．

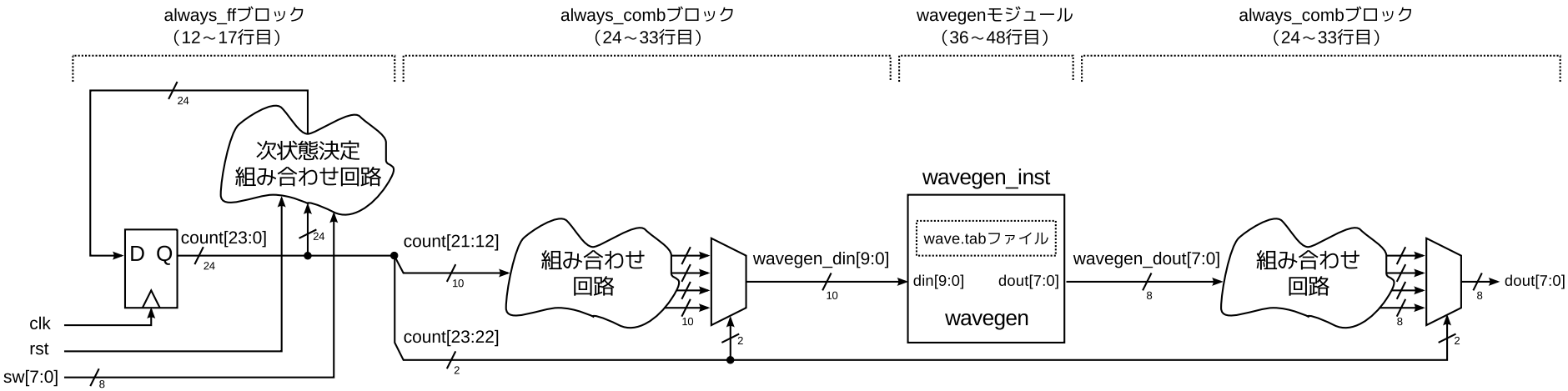
piano モジュール記述例 (1/2)

```
2  module piano
3    (
4     input wire          clk,
5     input wire          rst,
6     input wire [7:0]    sw,
7     output logic [7:0] dout
8     );
9
10    reg [23:0]          count;
11
12    always_ff @(posedge clk) begin
13      if (rst)
14        count <= 24'd0;
15      else
16        ...
17    end
18
19    logic [9:0] wavegen_din;
20    wire [7:0]  wavegen_dout;
21    wavegen wavegen_inst
22      (.din(wavegen_din), .dout(wavegen_dout));
23
24    always_comb begin
```

piano モジュール記述例 (2/2)

```
25        case (count[23:22])
26          2'b00: begin
27             wavegen_din <= count[21:12];
28             dout <= wavegen_dout;
29          end
30          2'b01:
31             ...
32        endcase
33     end
34 endmodule
35
36 module wavegen
37    (
38     input wire [9:0] din,
39     output reg [7:0] dout
40     );
41
42     always_comb begin
43        case (din)
44 `include "wave.tab"
45        endcase
46     end
47 endmodule
```

piano モジュールの構造



正弦波の生成

- 多項式展開： 四則演算で計算
  - Taylor 展開など
- 関数テーブル： あらかじめ計算して表にしておく
  - メモリによる実現
  - 組み合わせ回路による実現 ← 今回はこれ

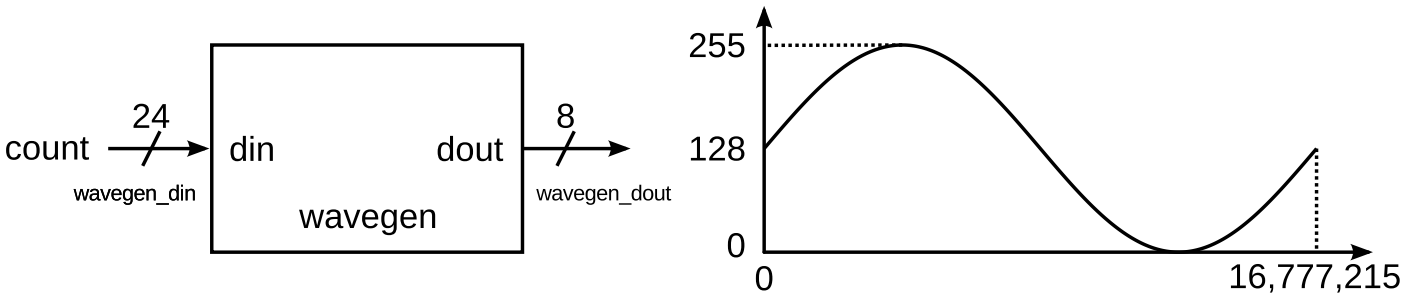
関数テーブルの実現法

(a) 24 入力 8 出力回路として実現
- 回路の複雑さが爆発

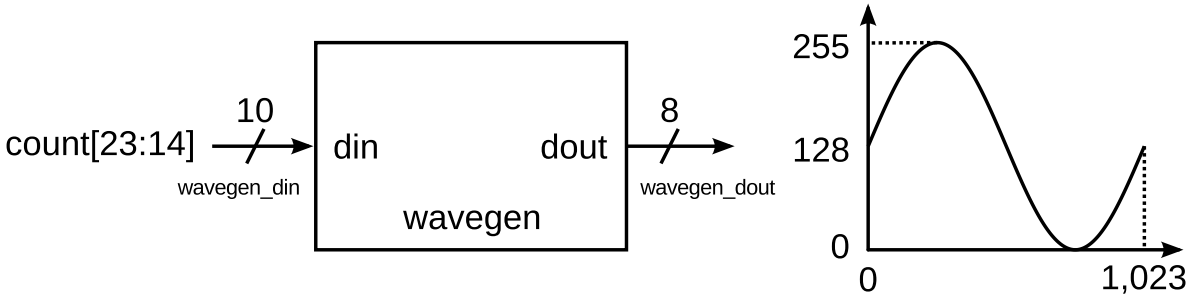
(b) 10 入力 8 出力回路として実現
- カウンタ値の下位 14 ビットは無視
- 「ぎざぎざ」の波形

(c) 第 1 象限のみを 10 入力 8 出力回路として実現 ← 今回はこれ
- 対称性を利用
- カウンタ値の下位 12 ビットを無視
- 上位 2 ビットで回路の出力値を変換
- 「ぎざぎざ」を緩和
- 24 行目〜32 行目の case 文で調整

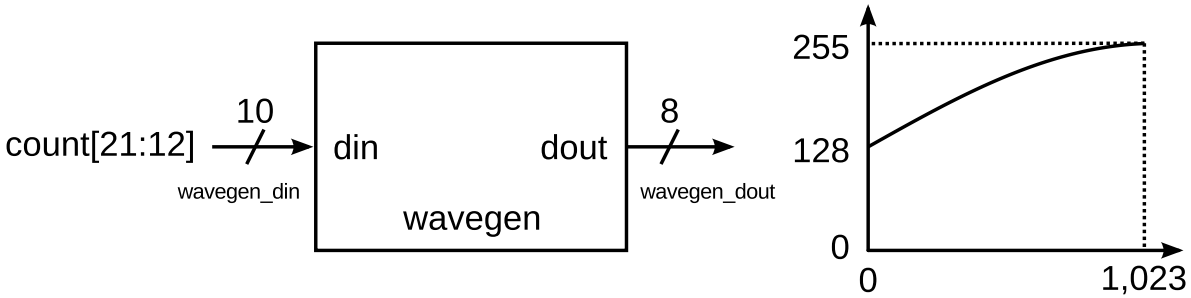
関数テーブルの実現法



(a) 24入力組み合わせ回路



(b) 10入力組み合わせ回路



(c) 第1象限のみの10入力組み合わせ回路

wavegen モジュールの記述例

```
36 module wavegen
37   (
38    input wire [9:0] din,
39    output reg [7:0] dout
40    );
41
42    always_comb begin
43       case (din)
44 `include "wave.tab"
45       endcase
46    end
47 endmodule
```

- 「`include」でファイルを include
- wave.tab ファイルの内容は C プログラムで自動生成

テーブルファイルの内容

```
10'd0: dout <= 8'd128;
10'd1: dout <= 8'd128;
  ...
```

- 関数値を計算し printf でこのように出力するプログラムを作ればよい

検証手順（1/3）

1. 関数テーブル作成
   - 関数値出力プログラム（wave.c）を作成
   - コンパイルしてテーブルファイル（wave.tab）に書き出し

```
% gcc -o wave -Wall -O2 wave.c -lm
% ./wave > wave.tab
```

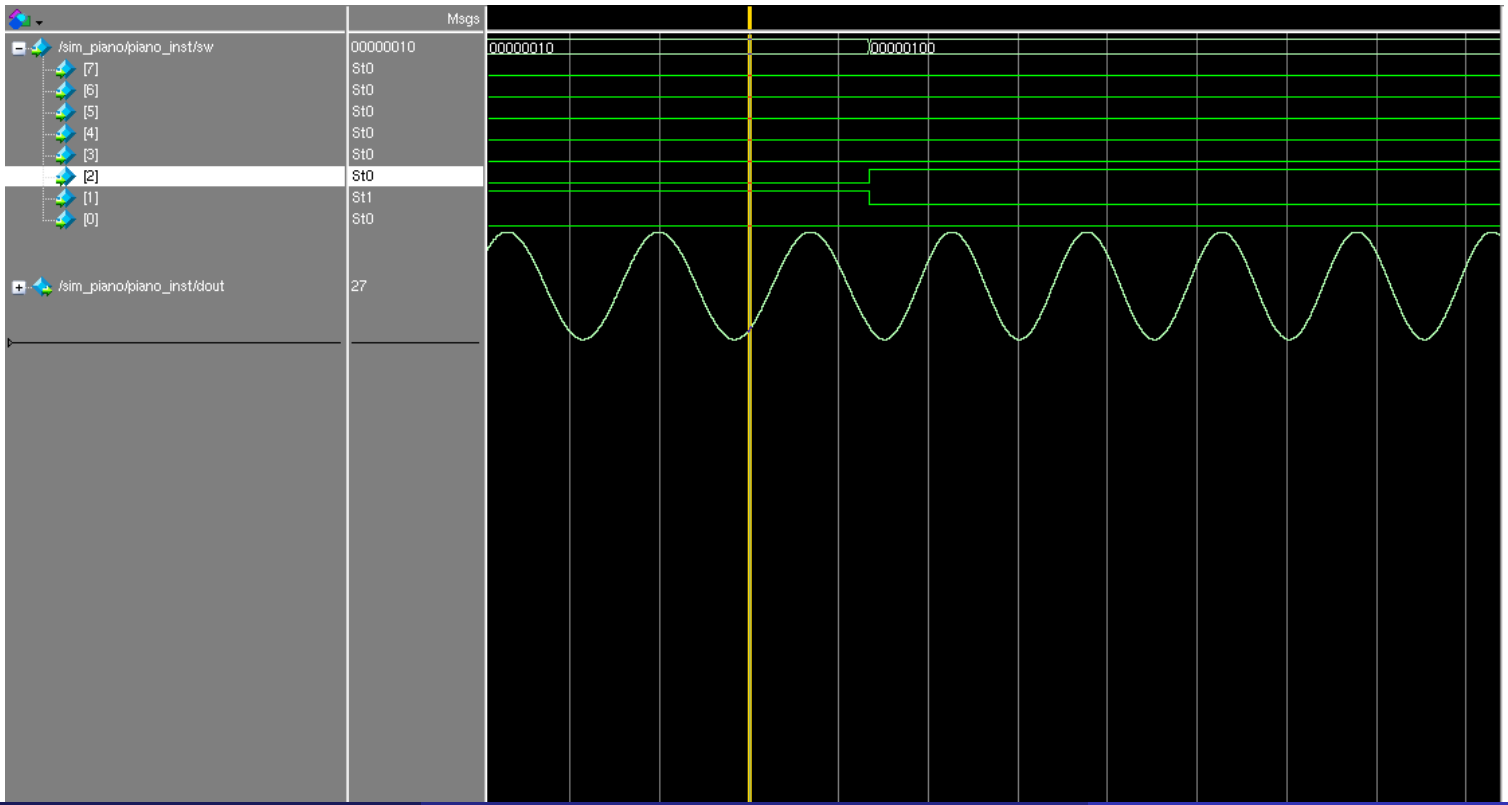
2. シミュレーション実行
   - 配布したシミュレーションモジュール（sim_piano.sv）を利用
   - スイッチボタンを左から順番に 50,000 クロックずつ押す動作

```
% make sim
```

検証手順（2/3）

3. 出力波形確認
   - 波形ビューアを起動
   - dout をアナログモードで表示し綺麗な正弦波出力か確認

```
% vsim sim_piano.wlf &
```



検証手順（3/3）

4. 配布 C プログラムによる検証（check_sim_piano.c）
   - シミュレーションログファイル（sim_piano.log）を読み込で出力波形を解析
   - 各スイッチの押下ごとに出力周波数を期待値と比較
   - 振幅が 0 から 255 まで振れているか，不連続変化がないかチェック
   - 周波数は出力が中央値（128）をクロスする時間間隔により算出

```
% make check
```

5. 実機確認
   - 論理合成・配置配線の後，FPGA にコンフィギュレーション
   - 複数のボタンを押したとき上の高い音が優先されているか確認

```
% make
% make config
```