附录

代码有删减，以方便打印。

基础组件

```
ENTITY SignalStablizer IS
  PORT(
    clk : IN STD_LOGIC;
    input : IN STD_LOGIC;
    output : OUT STD_LOGIC
  );
END;

ARCHITECTURE rtl OF SignalStablizer IS
  SIGNAL tmp : STD_LOGIC;
  SIGNAL counter : UNSIGNED(16 DOWNTO 0);
BEGIN
  PROCESS(clk) IS
  BEGIN
    IF RISING_EDGE(clk) THEN
      IF tmp=input THEN
        counter <= counter + 1;
      ELSE
        counter <= (OTHERS => '0');
        tmp <= input;
      END IF;
    END IF;
  END PROCESS;

  output <= tmp WHEN counter(16)='1';
END;

ENTITY Timer IS
  PORT(
    clk : IN STD_LOGIC;
    reset : IN STD_LOGIC;
    countdown : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    output : OUT STD_LOGIC
  );
END;

ARCHITECTURE rtl OF Timer IS
  SIGNAL cnt : UNSIGNED(16 DOWNTO 0);
BEGIN
  PROCESS(clk, reset) IS
  BEGIN
    IF reset='1' THEN
      cnt <= UNSIGNED('0' & countdown);
      output <= '0';
    ELSIF RISING_EDGE(clk) THEN
      IF cnt(16)='1' THEN
        cnt <= UNSIGNED('0' & countdown);
        output <= '1';
      ELSE
        cnt <= cnt-1;
        output <= '0';
```

```
      END IF;
    END IF;
  END PROCESS;
END;

-- Divides frequency by doubles of countdown
ENTITY ClockDivider IS
  PORT(
    clk : IN STD_LOGIC;
    countdown : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    output : OUT STD_LOGIC
  );
END;

ARCHITECTURE rtl OF ClockDivider IS
  SIGNAL cnt : UNSIGNED(16 DOWNTO 0);
  SIGNAL tmp : STD_LOGIC;
BEGIN
  PROCESS(clk) IS
  BEGIN
    IF RISING_EDGE(clk) THEN
      IF cnt(16)='1' THEN
        cnt <= UNSIGNED('0' & countdown);
        tmp <= NOT tmp;
      ELSE
        cnt <= cnt-1;
      END IF;
    END IF;
  END PROCESS;

  output <= tmp;
END;
```

逻辑控制模块

```
ENTITY GameController IS
  PORT(
    -- input clock
    clk    : IN STD_LOGIC;
    reset  : IN STD_LOGIC;
    -- input data
    start  : IN STD_LOGIC; -- if start key is pressed
    feed   : IN STD_LOGIC; -- if ready to feed a match result
    match  : IN STD_LOGIC; -- if input is a match
    -- stage information
    stage : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
    msg_length : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
    announcing_countdown : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    -- states
    idle     : OUT STD_LOGIC;
    stage_init   : OUT STD_LOGIC;
    announcing   : OUT STD_LOGIC;
    pending  : OUT STD_LOGIC
  );
END;

ARCHITECTURE rtl OF GameController IS
  -- NOTE to escape name with output signals
  TYPE State IS (State_Idle, State_StageInit, State_Announcing, State_Pending);

  SIGNAL next_state, current_state : State;
```

```
  SIGNAL next_stageid, current_stageid : UNSIGNED(1 DOWNTO 0);

  SIGNAL sec_threshold : UNSIGNED(3 DOWNTO 0);
  SIGNAL sec_counter : UNSIGNED(3 DOWNTO 0);
  SIGNAL tick_counter : UNSIGNED(7 DOWNTO 0);

  SIGNAL timer_reset : STD_LOGIC;
  SIGNAL tick_pulse : STD_LOGIC;
  SIGNAL sec_pulse : STD_LOGIC;
BEGIN
  PROCESS(clk, reset) IS
  BEGIN
    IF reset='1' THEN
      current_stageid <= "00";
      current_state <= State_Idle;
    ELSIF RISING_EDGE(clk) THEN
      current_stageid <= next_stageid;
      current_state <= next_state;
    END IF;
  END PROCESS;

  PROCESS(current_state, current_stageid, start, feed, sec_counter) IS
  BEGIN
    next_state <= current_state;
    next_stageid <= current_stageid;

    CASE current_state IS
      -- On welcome screen or success screen
      WHEN State_Idle =>
        IF start='1' THEN
          next_stageid <= "00";
          next_state <= State_StageInit;
        END IF;
      -- On notifying to initialize a new stage
      WHEN State_StageInit =>
        IF current_stageid="11" THEN
          next_state <= State_Idle;
        ELSE
          next_state <= State_Announcing;
        END IF;
      -- On announcing the random integer sequence
      WHEN State_Announcing =>
        IF sec_counter=sec_threshold THEN
          next_state <= State_Pending;
        END IF;
      -- On waiting for user input
      WHEN State_Pending =>
        IF feed='1' THEN
          next_state <= State_StageInit;

          IF match='1' THEN
            -- goto next stage
            next_stageid <= current_stageid+1;
          ELSE
            -- restart game session
            next_stageid <= "00";
          END IF;
        END IF;
      -- On failure
      WHEN OTHERS =>
```

```
      next_stageid <= "00";
      next_state <= State_Idle;
  END CASE;
END PROCESS;

-- Timers
T1 : Timer PORT MAP(
  clk => clk,
  reset => timer_reset,
  countdown => x"c350",
  output => tick_pulse
);

PROCESS(clk) IS
BEGIN
  IF RISING_EDGE(clk) THEN
    IF current_state=State_Announcing THEN
      IF tick_pulse='1' THEN
        tick_counter <= tick_counter+1;
      END IF;

      IF tick_counter=x"14" THEN
        tick_counter <= x"00";
        sec_counter <= sec_counter+1;
      END IF;
    ELSE
      tick_counter <= x"00";
      sec_counter <= x"0";
    END IF;
  END IF;
END PROCESS;

PROCESS(current_stageid) IS
BEGIN
  CASE current_stageid IS
    WHEN "00" =>
      msg_length <= o"5";
      sec_threshold <= x"5";
    WHEN "01" =>
      msg_length <= o"6";
      sec_threshold <= x"5";
    WHEN "10" =>
      msg_length <= o"6";
      sec_threshold <= x"3";
    WHEN OTHERS =>
      msg_length <= o"5";
      sec_threshold <= x"5";
  END CASE;
END PROCESS;

-- Passive internal signals
timer_reset <= '0' WHEN current_state=State_Announcing ELSE '1';

-- Passive output signals
stage <= STD_LOGIC_VECTOR(current_stageid);
announcing_countdown <= STD_LOGIC_VECTOR(sec_threshold-sec_counter);

idle <= '1' WHEN current_state=State_Idle ELSE '0';
stage_init <= '1' WHEN current_state=State_StageInit ELSE '0';
announcing <= '1' WHEN current_state=State_Announcing ELSE '0';
```

```
  pending <= '1' WHEN current_state=State_Pending ELSE '0';
END;
```

随机排列生成模块

```
-- PRNG via LFSR
-- https://en.wikipedia.org/wiki/Linear-feedback_shift_register
ENTITY RandGenerator IS
  PORT(
    clk    : IN  STD_LOGIC;
    reset  : IN  STD_LOGIC;
    enable : IN  STD_LOGIC;
    seed   : IN  STD_LOGIC_VECTOR(15 DOWNTO 0);
    data   : OUT STD_LOGIC
  );
END;

ARCHITECTURE rtl OF RandGenerator IS
  SIGNAL lfsr : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
  PROCESS(clk, reset) IS
    VARIABLE out_bit : STD_LOGIC := lfsr(0) XNOR lfsr(2) XNOR lfsr(3) XNOR lfsr(5);
  BEGIN
    IF reset = '1' THEN
      lfsr <= seed;
    ELSIF RISING_EDGE(clk) and enable='1' THEN
      lfsr <= out_bit & lfsr(15 DOWNTO 1);
    END IF;
  END PROCESS;

  -- in order to make full use of seed
  data <= lfsr(0);
END;

ENTITY PermBuffer IS
  PORT(
    clk    : IN  STD_LOGIC;
    reset  : IN  STD_LOGIC;
    flush  : IN  STD_LOGIC;
    ready  : OUT STD_LOGIC;
    data   : OUT STD_LOGIC_VECTOR(23 DOWNTO 0)
  );
END;

ARCHITECTURE rtl OF PermBuffer IS
  TYPE State IS (Void, Idle, Refreshing, Finalizing);

  SIGNAL next_state, current_state : State;

  -- counter used while refreshing the buffer
  SIGNAL counter : INTEGER RANGE 0 TO 15;
  -- buffer for 16-bit random number
  SIGNAL buf : STD_LOGIC_VECTOR(15 DOWNTO 0);
  -- output bit of random generator
  SIGNAL rand_bit : STD_LOGIC;
BEGIN
  PROCESS(clk, reset, flush) IS
  BEGIN
    IF reset='1' OR flush='1' THEN
      current_state <= Void;
    ELSIF RISING_EDGE(clk) THEN
```

```
      current_state <= next_state;

      -- timer on state Refreshing
      IF current_state=Refreshing THEN
        counter <= counter+1;
        buf <= rand_bit & buf(15 DOWNTO 1);
      ELSE
        counter <= 0;
      END IF;
    END IF;
  END PROCESS;

  PROCESS(current_state, counter) IS
  BEGIN
    -- default choice
    next_state <= current_state;

    CASE current_state IS
      -- when reset is taking place
      WHEN Void =>
        next_state <= Refreshing;
      -- when generator is giving away random bcd
      WHEN Idle =>
        NULL;
      -- when generator is refreshing random buffer
      WHEN Refreshing =>
        IF counter=15 THEN
          next_state <= Finalizing;
        END IF;
      -- when generator is loading bcd from random buffer
      WHEN Finalizing =>
        next_state <= Idle;
    END CASE;
  END PROCESS;

  PROCESS(buf) IS
    FUNCTION MakeBiasedBCD(v : STD_LOGIC_VECTOR(3 DOWNTO 0)) RETURN STD_LOGIC_VECTOR IS
      VARIABLE n : UNSIGNED(3 DOWNTO 0) := UNSIGNED(v);
    BEGIN
      IF(n < 10) THEN
        RETURN v;
      ELSE
        RETURN STD_LOGIC_VECTOR(n-10);
      END IF;
    END FUNCTION;
  BEGIN
    data <= MakeBiasedBCD(buf(3 DOWNTO 0))
        & MakeBiasedBCD(buf(6 DOWNTO 3))
        & MakeBiasedBCD(buf(9 DOWNTO 6))
        & MakeBiasedBCD(buf(12 DOWNTO 9))
        & MakeBiasedBCD(buf(15 DOWNTO 12))
        & MakeBiasedBCD(buf(12) & buf(9) & buf(6) & buf(3));
  END PROCESS;

  U: RandGenerator PORT MAP(clk, reset, '1', x"face", rand_bit);

  ready <= '1' WHEN current_state=Idle AND next_state=Idle ELSE '0';
END;
```

点阵显示模块

```
ENTITY LedArrayDriver IS
  PORT(
    clk : IN STD_LOGIC;
    disable : IN STD_LOGIC;
    img_code : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    output : OUT STD_LOGIC_VECTOR(23 DOWNTO 0)
  );
END;

ARCHITECTURE rtl OF LedArrayDriver IS
  ALIAS enable_out IS output(23 DOWNTO 16);
  ALIAS green_out IS output(15 DOWNTO 8);
  ALIAS red_out IS output(7 DOWNTO 0);

  SIGNAL frame_counter : UNSIGNED(2 DOWNTO 0);
  SIGNAL frame_enable : STD_LOGIC_VECTOR(7 DOWNTO 0);

  SIGNAL duty_counter : UNSIGNED(14 DOWNTO 0);
  SIGNAL color_threshold : UNSIGNED(14 DOWNTO 0);
  SIGNAL in_duty_cycle : STD_LOGIC;

  SIGNAL color_shift_enable : STD_LOGIC;
  SIGNAL red_enable : STD_LOGIC;
  SIGNAL green_enable : STD_LOGIC;

  SIGNAL red_bit: STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL green_bit: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
  PROCESS(disable, clk) IS
  BEGIN
    IF RISING_EDGE(clk) THEN
      IF disable='0' THEN
        duty_counter <= duty_counter+1;

        IF duty_counter(14)='1' THEN
          duty_counter(14) <= '0';
          frame_counter <= frame_counter+1;

          color_threshold <= color_threshold+1;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  U: LedArrayRom PORT MAP(
    clk => clk,
    disable => '0',
    code => img_code,
    frame => STD_LOGIC_VECTOR(frame_counter),
    red_bit => red_bit,
    green_bit => green_bit
  );

  WITH frame_counter SELECT
    frame_enable <= "11111110" WHEN o"0",
              "11111101" WHEN o"1",
              "11111011" WHEN o"2",
              "11110111" WHEN o"3",
```

```
                "11101111" WHEN o"4",
                "11011111" WHEN o"5",
                "10111111" WHEN o"6",
                "01111111" WHEN o"7";

  in_duty_cycle <= '1' WHEN duty_counter(13 DOWNTO 0)<color_threshold(13 DOWNTO 0) ELSE '0';

  color_shift_enable <= '1' WHEN img_code="000" ELSE '0';
  red_enable <= (NOT color_shift_enable) OR (color_threshold(14) XNOR in_duty_cycle);
  green_enable <= (NOT color_shift_enable) OR (color_threshold(14) XNOR (NOT in_duty_cycle));

  red_out <= red_bit WHEN red_enable='1' ELSE (OTHERS => '0');
  green_out <= green_bit WHEN green_enable='1' ELSE (OTHERS => '0');
  enable_out <= frame_enable;
END;

ENTITY LedArrayRom IS
  PORT(
    clk : IN STD_LOGIC;
    disable : IN STD_LOGIC;
    code : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    frame : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    red_bit : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
    green_bit : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END;

ARCHITECTURE rtl OF LedArrayRom IS
  CONSTANT data : STD_LOGIC_VECTOR(1023 DOWNTO 0)
    :=
"00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000"
    &
"00010000000000000011000000000000001000000000000001000000000000001000000000000001000000000000001000000000
00000010000000000000111000000000"
    &
"00111000000000000100010000000000000010000000000000100000000000001000000000000010000000000000100000000000
00000100000000000001111100000000"
    &
"00111000000000000100010000000000000010000000000000110000000000000010000000000000100000000000001000000000
00001000100000000000111000000000"
    &
"00000100000000000000110000000000001010000000000010010000000000100010000000000011111100000000
00000000100000000000000010000000"
    &
"01111100000000000100000000000000100000000000000111100000000000000010000000000000100000000000001000000000
00001000100000000000111000000000"
    &
"01010100010101000101010001010100001010000010100000000000000000011101001111010010100110101001
10101001011010010111110100111101001"
    &
"11101000111010001010101010101011000100110001001000001110000011011000000110000010001110100011
11010001010100010100110111001101110";

  SIGNAL index : INTEGER RANGE 0 TO 1023;
  SIGNAL current_line : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
  index <= TO_INTEGER(UNSIGNED(code & frame & "0000"));
  current_line <= data(index+15 DOWNTO index);
```

```
  red_bit <= current_line(15 DOWNTO 8) WHEN disable='0' ELSE (OTHERS => '0');
  green_bit <= current_line(7 DOWNTO 0) WHEN disable='0' ELSE (OTHERS => '0');
END;
```

数码管显示模块

```
ENTITY DigitArrayDriver IS
  PORT(
    clk : IN STD_LOGIC;
    disable : IN STD_LOGIC;
    input : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    output : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
  );
END;

ARCHITECTURE rtl OF DigitArrayDriver IS
  SIGNAL enable_out : STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL digit_out : STD_LOGIC_VECTOR(7 DOWNTO 0);

  SIGNAL index : INTEGER RANGE 0 TO 7;
  SIGNAL scan_pulse : STD_LOGIC;
BEGIN
  -- Clock
  PROCESS(clk, disable) IS
  BEGIN
    IF RISING_EDGE(clk) THEN
      IF scan_pulse='1' THEN
        index <= index+1;
      END IF;
    END IF;
  END PROCESS;

  -- Timer for scanning
  T1 : Timer PORT MAP(
    clk => clk,
    reset => disable,
    countdown => x"000f",
    output => scan_pulse
  );

  -- Passive output signals
  WITH input(index*4+3 DOWNTO index*4) SELECT
    digit_out <= x"3f" WHEN x"0",
          x"06" WHEN x"1",
          x"5b" WHEN x"2",
          x"4f" WHEN x"3",
          x"66" WHEN x"4",
          x"6d" WHEN x"5",
          x"7d" WHEN x"6",
          x"07" WHEN x"7",
          x"7f" WHEN x"8",
          x"6f" WHEN x"9",
          x"00" WHEN OTHERS;

  WITH index SELECT
    enable_out <= "11111110" WHEN 0,
           "11111101" WHEN 1,
           "11111011" WHEN 2,
           "11110111" WHEN 3,
           "11101111" WHEN 4,
           "11011111" WHEN 5,
```

```
            "10111111" WHEN 6,
            "01111111" WHEN 7;

  output <= enable_out&digit_out WHEN disable='0' ELSE x"0000";
END;
```

蜂鸣器模块

```
ENTITY AudioDriver IS
  PORT(
    clk : IN STD_LOGIC;
    enable : IN STD_LOGIC;
    output : OUT STD_LOGIC
  );
END;

ARCHITECTURE rtl OF AudioDriver IS
  SIGNAL disable : STD_LOGIC;

  SIGNAL beat_pulse : STD_LOGIC;
  SIGNAL beat_index : UNSIGNED(3 DOWNTO 0);
  SIGNAL cursor : INTEGER RANGE 0 TO 60;

  SIGNAL notes : STD_LOGIC_VECTOR(7 DOWNTO 0);

  SIGNAL tone : STD_LOGIC_VECTOR(2 DOWNTO 0);
  CONSTANT music : STD_LOGIC_VECTOR(179 DOWNTO 0)
    := o"33227722" & o"223333342222" &
       o"33227722" & o"334566544444" &
       o"33223277" & o"442244433333";
BEGIN
  disable <= NOT enable;
  T0: Timer PORT MAP(clk, disable, x"3d09", beat_pulse);

  U0: ClockDivider PORT MAP(clk, x"0001", notes(0));
  U1: ClockDivider PORT MAP(clk, x"03bc", notes(1));
  U2: ClockDivider PORT MAP(clk, x"0361", notes(2));
  U3: ClockDivider PORT MAP(clk, x"02f7", notes(3));
  U4: ClockDivider PORT MAP(clk, x"02cc", notes(4));
  U5: ClockDivider PORT MAP(clk, x"027e", notes(5));
  U6: ClockDivider PORT MAP(clk, x"0238", notes(6));
  U7: ClockDivider PORT MAP(clk, x"01fa", notes(7));

  PROCESS(clk, enable) IS
  BEGIN
    IF enable='0' THEN
      output <= '0';
    ELSIF RISING_EDGE(clk) THEN
      IF beat_pulse='1' THEN
        beat_index <= beat_index+1;

        IF beat_index(3)='1' THEN
          beat_index <= (OTHERS => '0');
          cursor <= cursor+1;

          IF cursor=60 THEN
            cursor <= 0;
          END IF;
        END IF;
      END IF;
```

```
      output <= notes(TO_INTEGER(UNSIGNED(tone)));
    END IF;
  END PROCESS;

  tone <= music(179-cursor*3 DOWNTO 177-cursor*3);
END;
```

键盘阵列驱动模块

```
ENTITY ButtonArrayDriver IS
  PORT(
    clk : IN STD_LOGIC;
    row : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    column : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    pulse : OUT STD_LOGIC;
    choice : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
  );
END;

ARCHITECTURE rtl OF ButtonArrayDriver IS
  TYPE State IS(Idle, Scanning, WaitRelease);

  SIGNAL next_state, current_state : State;

  SIGNAL column_counter : INTEGER RANGE 0 TO 3;

  SIGNAL column_index : INTEGER RANGE 0 TO 3;
  SIGNAL row_index : INTEGER RANGE 0 TO 3;

  SIGNAL do_scanning : STD_LOGIC;
  SIGNAL stop_scanning : STD_LOGIC;

  SIGNAL any_pressed_unsafe : STD_LOGIC;
  SIGNAL any_pressed : STD_LOGIC;
BEGIN
  PROCESS(clk) IS
  BEGIN
    IF RISING_EDGE(clk) THEN
      current_state <= next_state;

      IF next_state=Scanning THEN
        column_counter <= column_counter+1;

        -- put next column filter, not the current
        CASE column_counter IS
          WHEN 0 => column <= "1101";
          WHEN 1 => column <= "1011";
          WHEN 2 => column <= "0111";
          WHEN 3 => column <= "1110";
        END CASE;

        IF row(0)='0' THEN
          column_index <= column_counter;
          row_index <= 3;
        ELSIF row(1)='0' THEN
          column_index <= column_counter;
          row_index <= 2;
        ELSIF row(2)='0' THEN
          column_index <= column_counter;
```

```
          row_index <= 1;
        ELSIF row(3)='0' THEN
          column_index <= column_counter;
          row_index <= 0;
        END IF;

      ELSE
        column <= "0000";
      END IF;
    END IF;
  END PROCESS;

  PROCESS(current_state, any_pressed, stop_scanning) IS
  BEGIN
    pulse <= '0';
    next_state <= current_state;

    CASE current_state IS
      WHEN Idle =>
        IF any_pressed='1' THEN
          next_state <= Scanning;
        END IF;
      WHEN Scanning =>
        IF stop_scanning='1' THEN
          pulse <= '1';
          next_state <= WaitRelease;
        END IF;
      WHEN WaitRelease =>
        IF any_pressed='0' THEN
          next_state <= Idle;
        END IF;
    END CASE;
  END PROCESS;

  -- scanning control
  do_scanning <= '0' WHEN current_state=Scanning ELSE '1';

  T1 : Timer PORT MAP(
    clk => clk,
    reset => do_scanning,
    countdown => x"00ff",
    output => stop_scanning
  );

  any_pressed_unsafe <= NOT (row(0) AND row(1) AND row(2) AND row(3));
  U: SignalStablizer PORT MAP(clk, any_pressed_unsafe, any_pressed);

  -- passive output signals
  choice <= STD_LOGIC_VECTOR(TO_UNSIGNED(row_index*4+column_index, 4));
END;
```

排序模块

```
ENTITY MessageRearranger IS
  PORT(
    clk : IN STD_LOGIC;
    input : IN STD_LOGIC_VECTOR(23 DOWNTO 0);
    output : OUT STD_LOGIC_VECTOR(23 DOWNTO 0)
  );
END;
```

```
ARCHITECTURE rtl OF MessageRearranger IS
  SIGNAL buf : STD_LOGIC_VECTOR(23 DOWNTO 0);

  SIGNAL flag : STD_LOGIC;
  SIGNAL tmp : STD_LOGIC_VECTOR(23 DOWNTO 0);
BEGIN
  PROCESS(clk) IS
    FUNCTION SortBlock(v : STD_LOGIC_VECTOR(7 DOWNTO 0)) RETURN STD_LOGIC_VECTOR IS
    BEGIN
      IF(UNSIGNED(v(7 DOWNTO 4)) < UNSIGNED(v(3 DOWNTO 0))) THEN
        RETURN v;
      ELSE
        RETURN v(3 DOWNTO 0) & v(7 DOWNTO 4);
      END IF;
    END FUNCTION;

    FUNCTION SortEach(v : STD_LOGIC_VECTOR(23 DOWNTO 0)) RETURN STD_LOGIC_VECTOR IS
    BEGIN
      RETURN SortBlock(v(23 DOWNTO 16)) &
             SortBlock(v(15 DOWNTO 8)) &
             SortBlock(v(7 DOWNTO 0));
    END FUNCTION;

    FUNCTION SortCenter(v : STD_LOGIC_VECTOR(23 DOWNTO 0)) RETURN STD_LOGIC_VECTOR IS
    BEGIN
      RETURN v(23 DOWNTO 20) &
             SortBlock(v(19 DOWNTO 12)) &
             SortBlock(v(11 DOWNTO 4)) &
             v(3 DOWNTO 0);
    END FUNCTION;
  BEGIN
    IF RISING_EDGE(clk) THEN
      IF buf=input THEN
        flag <= NOT flag;

        IF flag='1' THEN
          tmp <= SortEach(tmp);
        ELSE
          tmp <= SortCenter(tmp);
        END IF;
      ELSE
        buf <= input;
        tmp <= input;
      END IF;
    END IF;
  END PROCESS;

  output <= tmp;
END;
```

全局调度模块

```
ENTITY Device IS
  PORT(
    clk : IN STD_LOGIC; -- global clock

    swt_hard : IN STD_LOGIC;
    btn_reset : IN STD_LOGIC;
    btn_start : IN STD_LOGIC;
    btn_undo : IN STD_LOGIC;
```

```
    btn_array_row : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    btn_array_col : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);

    digit_array : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
    led_array : OUT STD_LOGIC_VECTOR(23 DOWNTO 0);
    beeper_on : OUT STD_LOGIC
  );
END;

ARCHITECTURE rtl OF Device IS
  -- control signals
  SIGNAL inner_clk : STD_LOGIC;
  SIGNAL cmd_reset, cmd_start, cmd_undo : STD_LOGIC;

  -- msg generation and input
  SIGNAL msg_ready : STD_LOGIC;
  SIGNAL msg_data_raw : STD_LOGIC_VECTOR(23 DOWNTO 0);
  SIGNAL msg_data : STD_LOGIC_VECTOR(23 DOWNTO 0);
  SIGNAL msg_data_rearranged : STD_LOGIC_VECTOR(23 DOWNTO 0);
  SIGNAL msg_answer : STD_LOGIC_VECTOR(23 DOWNTO 0);

  SIGNAL msg_in_pulse : STD_LOGIC;
  SIGNAL msg_in_buffer : STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL msg_index : INTEGER RANGE 0 TO 7;
  SIGNAL msg_input : STD_LOGIC_VECTOR(23 DOWNTO 0);

  -- game information
  SIGNAL idle, stage_init, announcing, pending : STD_LOGIC;
  SIGNAL stage_id : STD_LOGIC_VECTOR(1 DOWNTO 0);
  SIGNAL msg_length : STD_LOGIC_VECTOR(2 DOWNTO 0);
  SIGNAL announcing_countdown : STD_LOGIC_VECTOR(3 DOWNTO 0);

  -- additional game information
  SIGNAL win : STD_LOGIC;
  SIGNAL cd_img : STD_LOGIC_VECTOR(2 DOWNTO 0);
  SIGNAL msg_mask : STD_LOGIC_VECTOR(23 DOWNTO 0);

  SIGNAL input_fit : STD_LOGIC;
  SIGNAL input_match : STD_LOGIC;

  -- output buffer
  SIGNAL digit_data : STD_LOGIC_VECTOR(31 DOWNTO 0); -- 4*8
  SIGNAL screen_data : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
  -- in 50MHz divided by a factor of 2*25, out 1MHz
  U0 : ClockDivider PORT MAP(clk, x"0019", inner_clk);

  U1 : SignalStablizer PORT MAP(inner_clk, btn_reset, cmd_reset);
  U2 : SignalStablizer PORT MAP(inner_clk, btn_start, cmd_start);
  U3 : SignalStablizer PORT MAP(inner_clk, btn_undo, cmd_undo);

  PROCESS(inner_clk) IS
  BEGIN
    IF RISING_EDGE(inner_clk) THEN
      IF cmd_undo='0' AND pending='1' THEN
        IF input_fit='0' AND msg_in_pulse='1' THEN
          IF UNSIGNED(msg_in_buffer) < "1010" THEN
            msg_index <= msg_index+1;
            msg_input(23-msg_index*4 DOWNTO 20-msg_index*4) <= msg_in_buffer;
```

```
        END IF;
      END IF;
    ELSE
      msg_index <= 0;
      msg_input <= x"ffffff";
    END IF;

    IF swt_hard='1' THEN
      msg_answer <= msg_data_rearranged;
    ELSE
      msg_answer <= msg_data;
    END IF;
  END IF;
END PROCESS;

U4 : GameController PORT MAP(
  clk => inner_clk,
  reset => cmd_reset,

  start => cmd_start,
  feed => input_fit,
  match => input_match,

  stage => stage_id,
  msg_length => msg_length,
  announcing_countdown => announcing_countdown,

  idle => idle,
  stage_init => stage_init,
  announcing => announcing,
  pending => pending
);

U5 : PermBuffer PORT MAP(
  clk => inner_clk,
  reset => '0',
  flush => stage_init,

  ready => msg_ready,
  data => msg_data_raw
);

U6 : LedArrayDriver PORT MAP(
  clk => clk,
  disable => '0',

  img_code => screen_data,
  output => led_array
);

U7 : DigitArrayDriver PORT MAP(
  clk => clk,
  disable => '0',

  input => digit_data,
  output => digit_array
);

U8 : AudioDriver PORT MAP(
  clk => inner_clk,
```

```
    enable => win,

    output => beeper_on
  );

  U9 : ButtonArrayDriver PORT MAP(
    clk => inner_clk,

    row => btn_array_row,
    column => btn_array_col,

    pulse => msg_in_pulse,
    choice => msg_in_buffer
  );

  U10 : MessageRearranger PORT MAP(
    clk => inner_clk,
    input => msg_data,
    output => msg_data_rearranged
  );

  win <= '1' WHEN idle='1' AND stage_id="11" ELSE '0';

  WITH announcing_countdown SELECT
    cd_img <= o"6" WHEN x"1",
          o"5" WHEN x"2",
          o"4" WHEN x"3",
          o"3" WHEN x"4",
          o"2" WHEN x"5",
          o"7" WHEN OTHERS;

  WITH msg_length SELECT
    msg_mask <= x"f00000" WHEN o"1",
            x"ff0000" WHEN o"2",
            x"fff000" WHEN o"3",
            x"ffff00" WHEN o"4",
            x"fffff0" WHEN o"5",
            x"ffffff" WHEN o"6",
            x"ffffff" WHEN OTHERS; -- invalid

  msg_data <= msg_data_raw OR (NOT msg_mask);

  input_fit <= '1' WHEN msg_length=STD_LOGIC_VECTOR(TO_UNSIGNED(msg_index, 3)) ELSE '0';
  input_match <= '1' WHEN (msg_answer AND msg_mask)=(msg_input AND msg_mask) ELSE '0';

  screen_data <= o"1" WHEN idle='1' AND stage_id="11" ELSE
           o"0" WHEN idle='1' AND stage_id="00" ELSE
           cd_img WHEN announcing='1' ELSE
           o"7";

  digit_data(31 DOWNTO 8) <= msg_data WHEN announcing='1' ELSE msg_input;
  digit_data(7 DOWNTO 4) <= x"f";
  digit_data(3 DOWNTO 0) <= "00" & stage_id WHEN idle='0' ELSE x"f";
END;
```