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)   
 := "00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000"  
 & "00010000000000000011000000000000000100000000000000010000000000000001000000000000000100000000000000010000000000000011100000000000"  
 & "00111000000000000100010000000000000001000000000000000100000000000000100000000000000100000000000000100000000000000111110000000000"  
 & "00111000000000000100010000000000000001000000000000011000000000000000010000000000000001000000000001000100000000000011100000000000"  
 & "00000100000000000000110000000000000101000000000000100100000000000100010000000000011111100000000000000100000000000000010000000000"  
 & "01111100000000000100000000000000010000000000000001111000000000000000010000000000000001000000000001000100000000000011100000000000"  
 & "01010100010101000101010001010100001010000010100000000000000000001110100111101001010011010100110101001011010010111110100111101001"  
 & "11101000111010001010101010101010110001001100010010000011100000110110000001100000100011101000111010001010100010100110111001101110";  
  
 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;