

# Compulsory Assignment: Music Box

Victor J Hansen<sup>1</sup>    Devidas Kazokas<sup>2</sup>  
Leila Mozaffari<sup>3</sup>    Biplav Karna<sup>4</sup>

USN Kongsberg

Software/Hardware co-development of Embedded Systems

# Overview

- 1 Problem Description
- 2 FSM Block Diagram
- 3 ASMD Diagram
- 4 Code Walkthrough
  - VHDL Design Walkthrough
  - Vivado Schematic Diagram
  - FSM Vhdl Implementation
- 5 Simulation
  - Simulation: TestBench Snippet
  - Output
- 6 Thanks

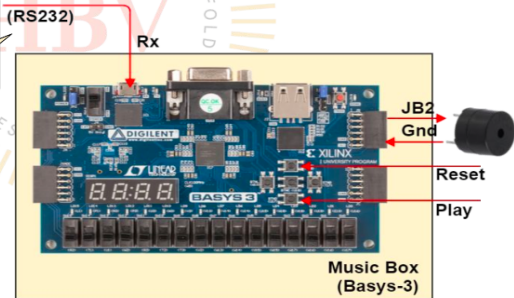
## Overview of the music box

Check section 11.4 of P. Chu's book

- Store the melody a local RAM as a sequence of ASCII characters typed in the computer keyboard
- Press “Play” to listen



e.g. PuTTY or Hyperterminal

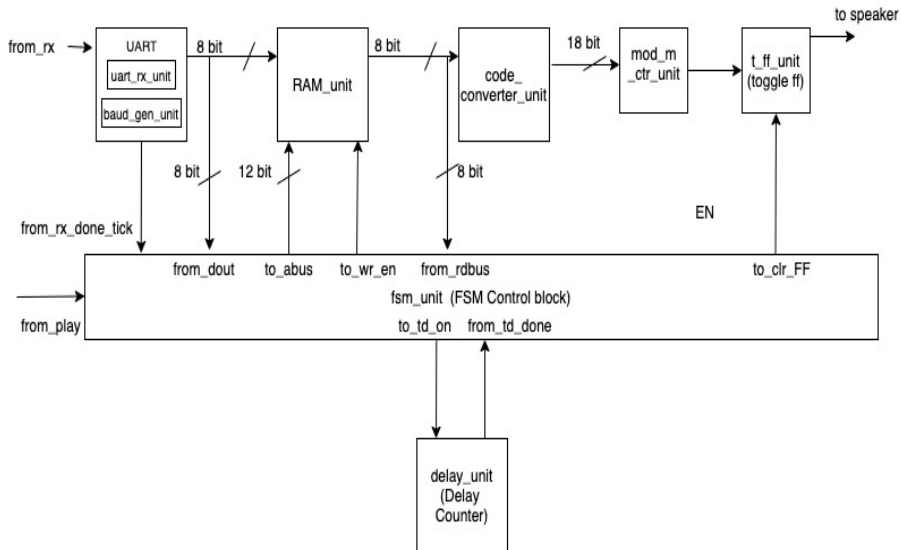


# Music Box: Assumptions

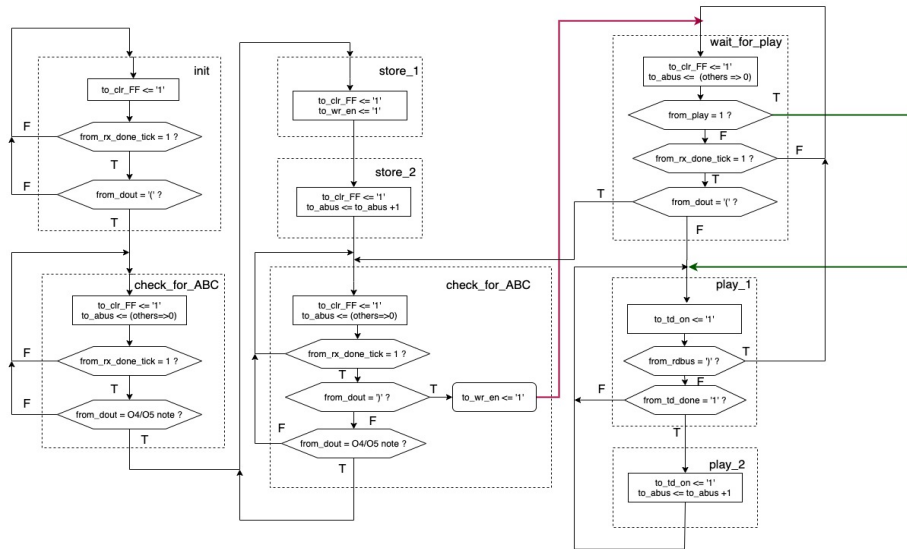
We have assumed below things for design of the music box.

- Only major notes of octaves 4 and 5 are taken into consideration.
- The duration of playing note is fixed to 0.5 sec.
- The user has to start music with '(' and end with ')'.  
(Note: The original image contains a typo 'The user has to start music with ' and end with ')'. This has been corrected to '(' and ')' for accuracy.)
- Repeated '(' and ')' will be ignored.
- Only one music can be stored.
- The user cannot provide new music when playing.
- The user can play stored music multiple times.

# FSM Block Diagram



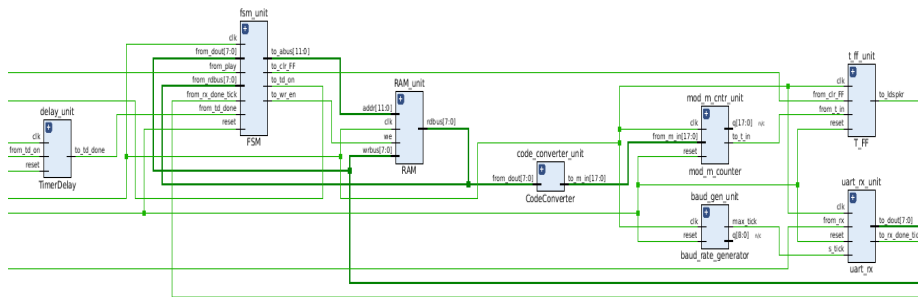
# ASMD Diagram



## VHDL Design Walkthrough: Hierarchical

- top.vhd
  - uart\_rx.vhd
    - baud\_rate\_generator.vhd
  - FSM.vhd
  - RAM.vhd
  - code\_converter.vhd
  - mod\_m\_cntr.vhd
  - t\_ff.vhd
  - timer\_delay.vhd

# Vivado Schematic Diagram





# Snippet of FSM.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity FSM is
    generic (
        ADDR_WIDTH: integer := 12;
        DATA_WIDTH: integer := 8 );

    port (
        clk, reset, from_play : in STD_LOGIC;
        from_rx_done_tick      : in STD_LOGIC;
        from_td_done           : in STD_LOGIC;
        from_dout               : in STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
        from_rdbus              : in STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
        to_abus                 : out STD_LOGIC_VECTOR (ADDR_WIDTH-1 downto 0);
        to_wr_en                : out STD_LOGIC;
        to_td_on                : out STD_LOGIC;
        to_clr_FF               : out STD_LOGIC );
end FSM;

architecture arch of FSM is
    type state_type is (init, check_for_ABC, store_1, store_2, store_3,
        wait_for_play, play_1, play_2);
    signal state_next, state_reg : state_type;
    signal pcntr_next, pcntr_reg : unsigned (ADDR_WIDTH-1 downto 0); — program counter (
        increment address register)
begin
    — state register
    process(clk, reset) begin
        if (reset = '1') then
            state_reg <= init;
            pcntr_reg <= (others => '0');
        elsif rising_edge(clk) then
            state_reg <= state_next;
            pcntr_reg <= pcntr_next;
        end if;
    end process;
end arch;
```

```

    end if;
end process;

```

— next state and output logic

```

process(from_play, state_reg, pcntr_reg, from_rx_done_tick, from_dout, from_rdbus,
        from_td_done)

```

```

begin

```

```

    state_next <= state_reg;
    pcntr_next <= pcntr_reg; — address counter
    to_clr_FF <= '0';
    to_td_on <= '0';
    to_wr_en <= '0';
    case state_reg is

```

---

```

        when init =>
            to_clr_FF <= '1';
            if (from_rx_done_tick = '1') then
                if (from_dout = X"28") then — ASCII for '('
                    state_next <= check_for_ABC;
                end if;
            end if;

```

---

```

        when check_for_ABC =>
            to_clr_FF <= '1';
            pcntr_next <= (others => '0');
            if (from_rx_done_tick = '1') then
                — octave 4 (could this be done in a simpler way)
                if ( from_dout = X"43" or from_dout = X"44" or from_dout = X"45" or
                    from_dout = X"46"
                    or from_dout = X"47" or from_dout = X"41" or from_dout = X"42"
                    — octave 5
                    or from_dout = X"63" or from_dout = X"64" or from_dout = X"65" or
                    from_dout = X"66"
                    or from_dout = X"67" or from_dout = X"61" or from_dout = X"62") then
                    state_next <= store_1;
                end if;

```

```
end if;
```

---

```
when store_1 =>  
  to_clr_FF <= '1';  
  to_wr_en <= '1';  
  state_next <= store_2;
```

---

```
when store_2 =>  
  to_clr_FF <= '1';  
  pcntr_next <= pcntr_reg + 1;  
  state_next <= store_3;
```

---

```
when store_3 =>  
  to_clr_FF <= '1';  
  if (from_rx_done_tick = '1') then  
    if (from_dout = X"29") then      — ASCII for ')''  
      to_wr_en <= '1';  
      state_next <= wait_for_play;  
      — octave 4  
    elsif (from_dout = X"43" or from_dout = X"44" or from_dout = X"45" or  
           from_dout = X"46"  
           or from_dout = X"47" or from_dout = X"41" or from_dout = X"42"  
           — octave 5  
           or from_dout = X"63" or from_dout = X"64" or from_dout = X"65" or  
           from_dout = X"66"  
           or from_dout = X"67" or from_dout = X"61" or from_dout = X"62") then  
      state_next <= store_1;  
    end if;  
  end if;
```

---

```
when wait_for_play =>  
  to_clr_FF <= '1';  
  pcntr_next <= (others => '0');  
  if (from_play = '1') then  
    state_next <= play_1;  
  elsif (from_rx_done_tick = '1') then
```

```
        if (from_dout = X"28") then — — ASCII for '('
            state_next <= check_for_ABC;
        end if;
    end if;
```

---

```
when play_1 =>
    to_td_on <= '1';
    if (from_rdbus = X"29") then — ')' end of tune
        state_next <= wait_for_play;
    elsif (from_td_done = '1') then
        state_next <= play_2;
    end if;
```

---

```
when play_2 =>
    to_td_on <= '1';
    pcntr_next <= pcntr_reg + 1;
    state_next <= play_1;
end case;
end process;
to_abus <= std_logic_vector (pcntr_reg);
end arch;
```

# Snippet of testbench,vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY musicbox_tb IS
END musicbox_tb;

ARCHITECTURE behavior OF musicbox_tb IS
    COMPONENT top
        port (
            rx, clk, reset, play : IN std_logic;
            loudspeaker : OUT std_logic;
            leds : OUT std_logic_vector(7 downto 0) );
    END COMPONENT;

    signal rx, clk, reset, play : std_logic := '0';
    signal loudspeaker : std_logic;
    signal leds : std_logic_vector(7 downto 0);

    constant clk_period : time := 10 ns;
    constant bit_period : time := 52083ns; — time for 1 bit.. 1bit/19200bps = 52.08 us
    constant rx_data_ascii_B4: std_logic_vector(7 downto 0) := "01000010"; — send Tone
        B4
    constant rx_data_ascii_B5: std_logic_vector(7 downto 0) := "01100010"; — send Tone
        B5
    constant rx_data_ascii_C4: std_logic_vector(7 downto 0) := "01000011"; — send Tone
        C4
    constant rx_data_ascii_F4: std_logic_vector(7 downto 0) := "01000110"; — send Tone
        F4
    constant start_tune: std_logic_vector(7 downto 0) := "00101000"; — start of tune '('
    constant end_tune: std_logic_vector(7 downto 0) := "00101001"; — end of tune ')'

BEGIN
    uut: top PORT MAP
        ( rx => rx,
          clk => clk,
```

```

        reset => reset ,
        loudspeaker => loudspeaker ,
        play => play ,
    leds => leds );

```

```

clk_process: process begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
end process;

```

```

stim_proc: process begin
    reset <= '1';
    play <= '0';
    wait for 100 ns;
    reset <= '0';
    rx <= '1';
    wait for clk_period;

    — Send start of tune '('
    rx <= '0';          — start bit
    wait for bit_period;
    for i in 0 to 7 loop
        rx <= start_tune(i); — 8 data bits
        wait for bit_period;
    end loop;
    rx <= '1'; — stop bit
    wait for bit_period;
    rx <= '1'; — idle
    wait for clk_period*10;

```

```

— Test Tone B5
rx <= '0'; — start bit
wait for bit_period;

```

```

for i in 0 to 7 loop
    rx <= rx_data_ascii-B5(i);
    wait for bit_period;
end loop;
rx <= '1'; — stop bit
wait for bit_period;
rx <= '1'; — idle
wait for clk_period*10;

```

— *Test Tone C4*

```

rx <= '0'; — start bit
wait for bit_period;
for i in 0 to 7 loop
    rx <= rx_data_ascii-C4(i);
    wait for bit_period;
end loop;
rx <= '1'; — stop bit
wait for bit_period;
rx <= '1'; — idle
wait for clk_period*10;

```

— *Test Tone F4*

```

rx <= '0'; — start bit
wait for bit_period;
for i in 0 to 7 loop
    rx <= rx_data_ascii-F4(i);
    wait for bit_period;
end loop;
rx <= '1'; — stop bit
wait for bit_period;
rx <= '1'; — idle
wait for clk_period*10;

```

```

— Send end of tune ')'
rx <= '0'; — start bit
wait for bit_period;
for i in 0 to 7 loop
    rx <= end_tune(i);
    wait for bit_period;
end loop;
rx <= '1'; — stop bit
wait for bit_period;
rx <= '1'; — idle
wait for clk_period*10;

```

```

play <= '1';
wait for clk_period*10;
play <= '0';
wait for clk_period*1000000;
play <= '1';
wait for clk_period*1000000;
play <= '0';
wait for clk_period;
wait;

```

```

end process;

```

```

END;

```





Thanks

*Thank You*