Appendices

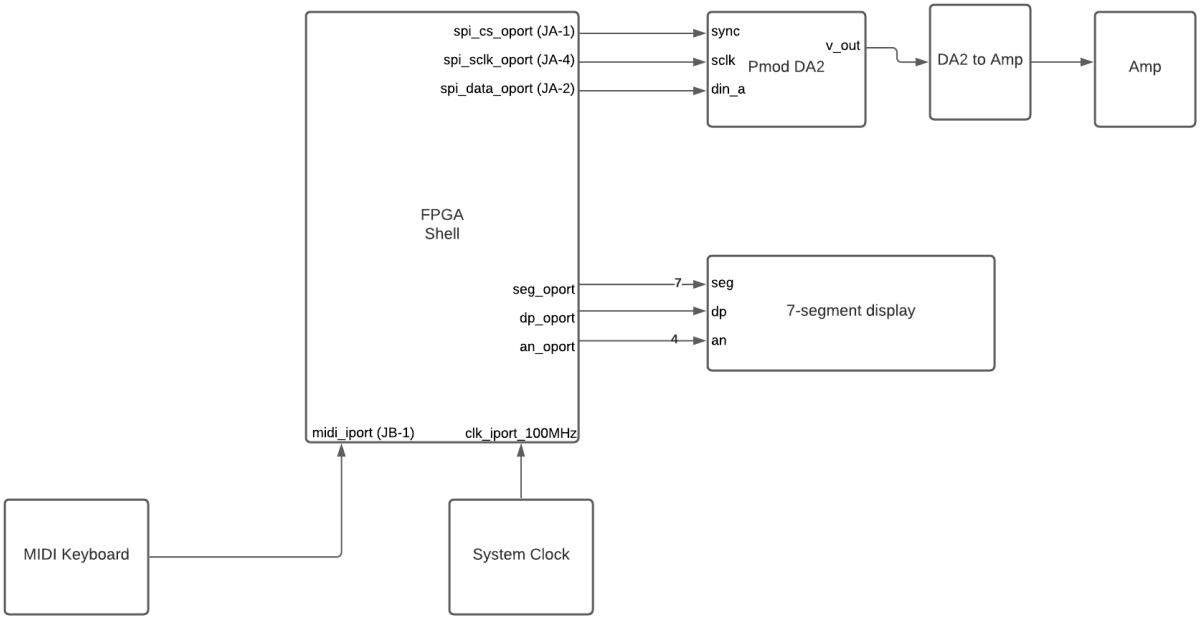
Appendix 1: System Level Diagrams

1.1: Front Panel

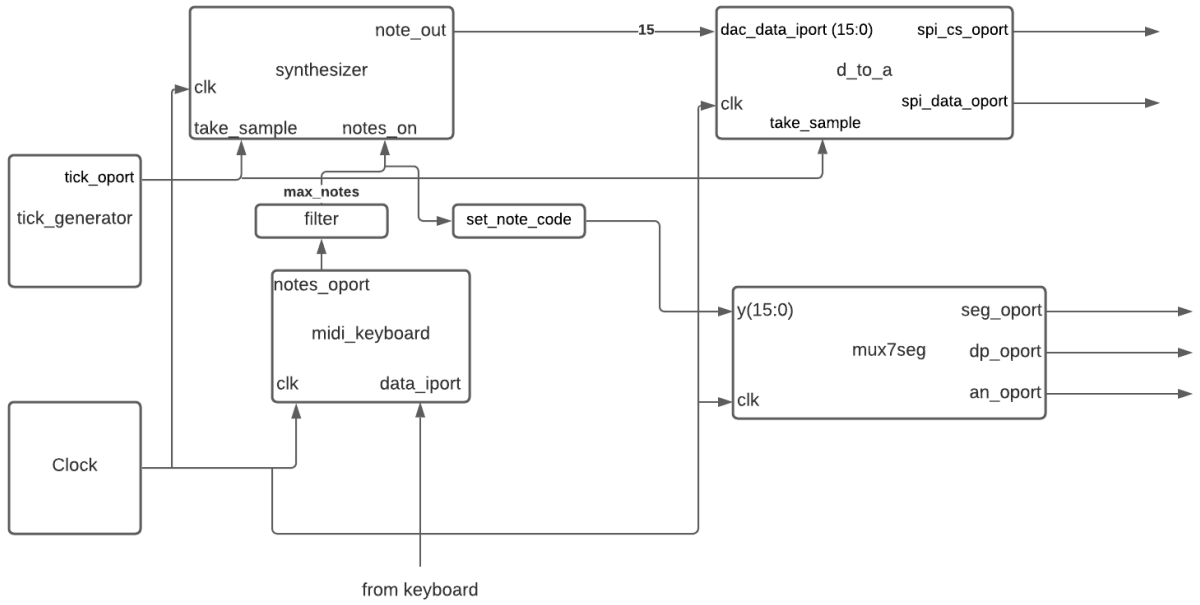
1. Basys 3 board
2. 7-segment LED display. When notes are pressed, the display updates to show the MIDI code (in hex) of the *highest* note pressed.
3. Pmod DA2 (digital-to-analog converter)
4. Pmod DA2 to AMP2 adapter
5. Pmod AMP2
6. MIDI to Pmod adapter (MIDI in)
7. Alesis Q25 MIDI keyboard
8. Power in

1.2: Functional Block Diagrams

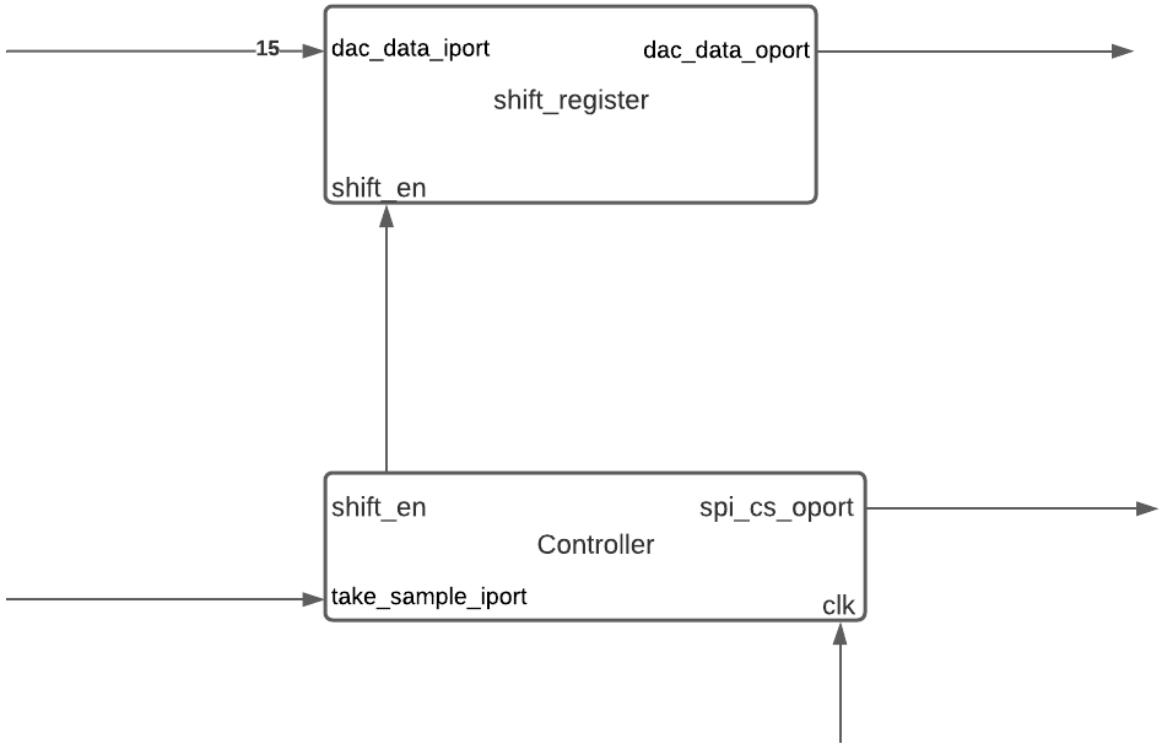
1.2.1: Top Level (Functional)



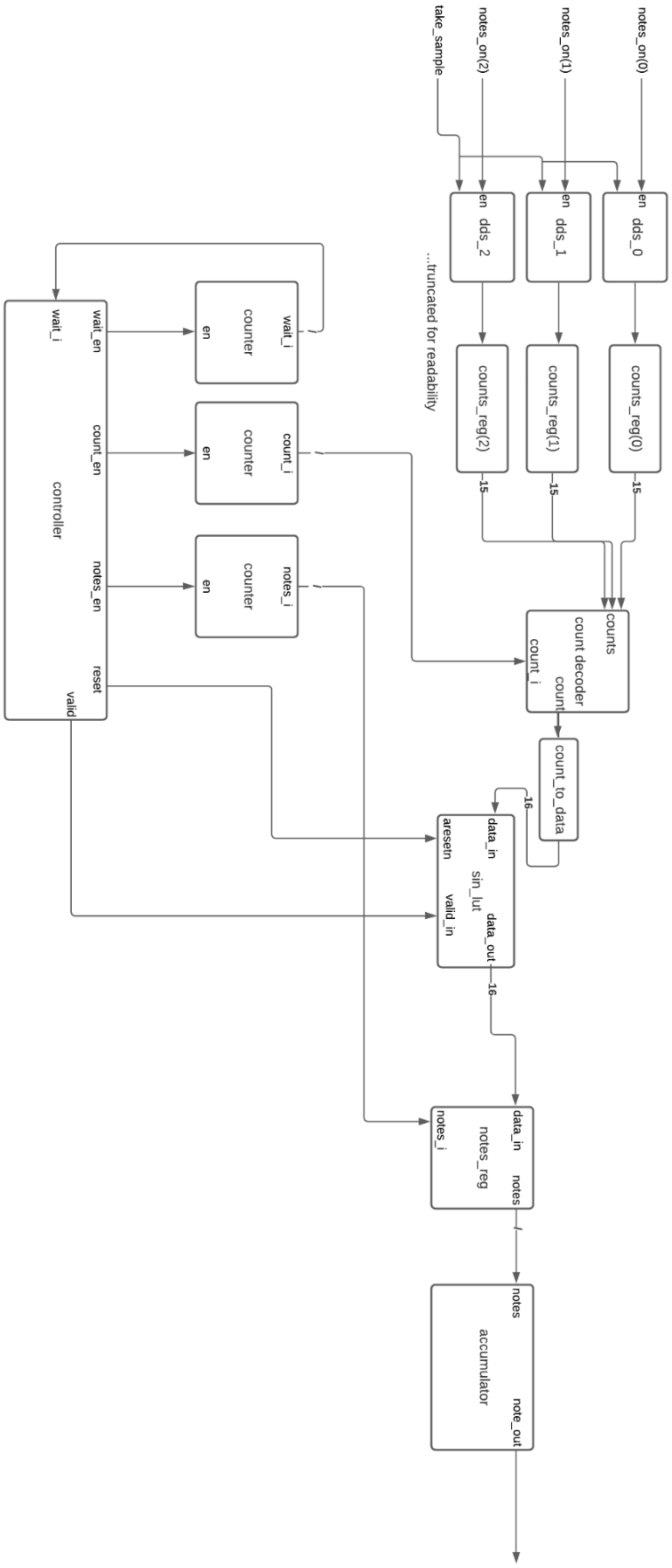
1.2.2: Top Level (FPGA)



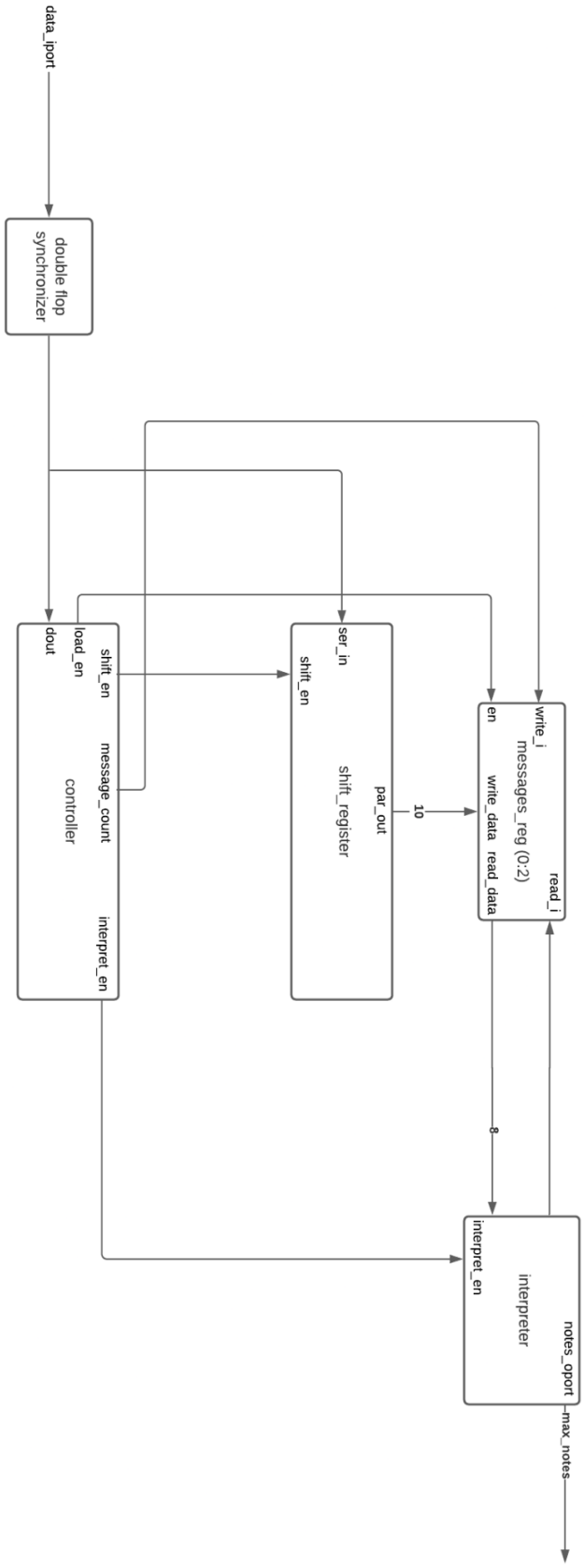
1.2.3: Digital-to-Analog Converter



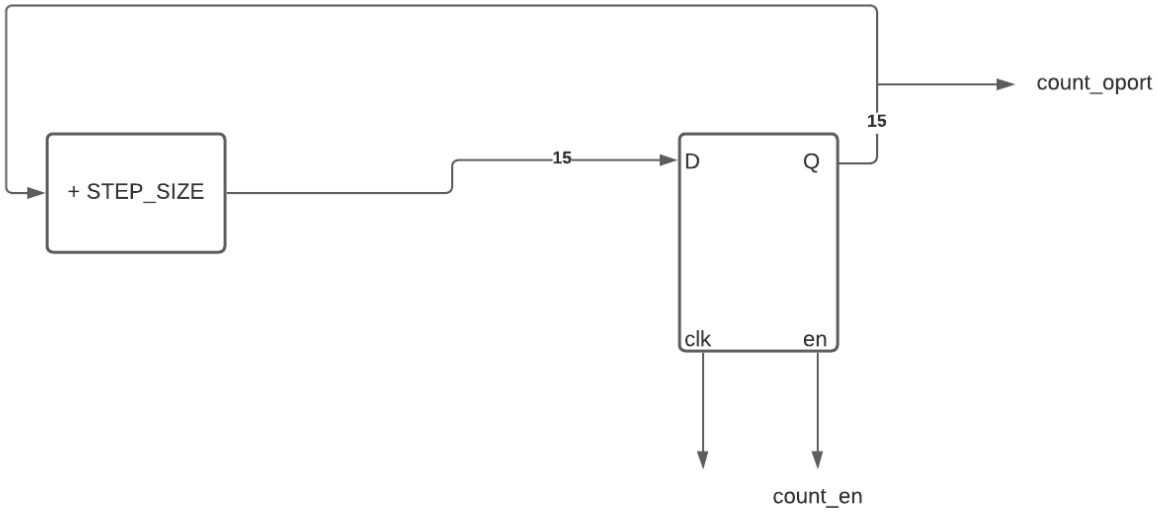
1.2.4: Note Synthesizer



1.2.5: MIDI Keyboard Interface



1.2.6: DDS Phase Accumulator



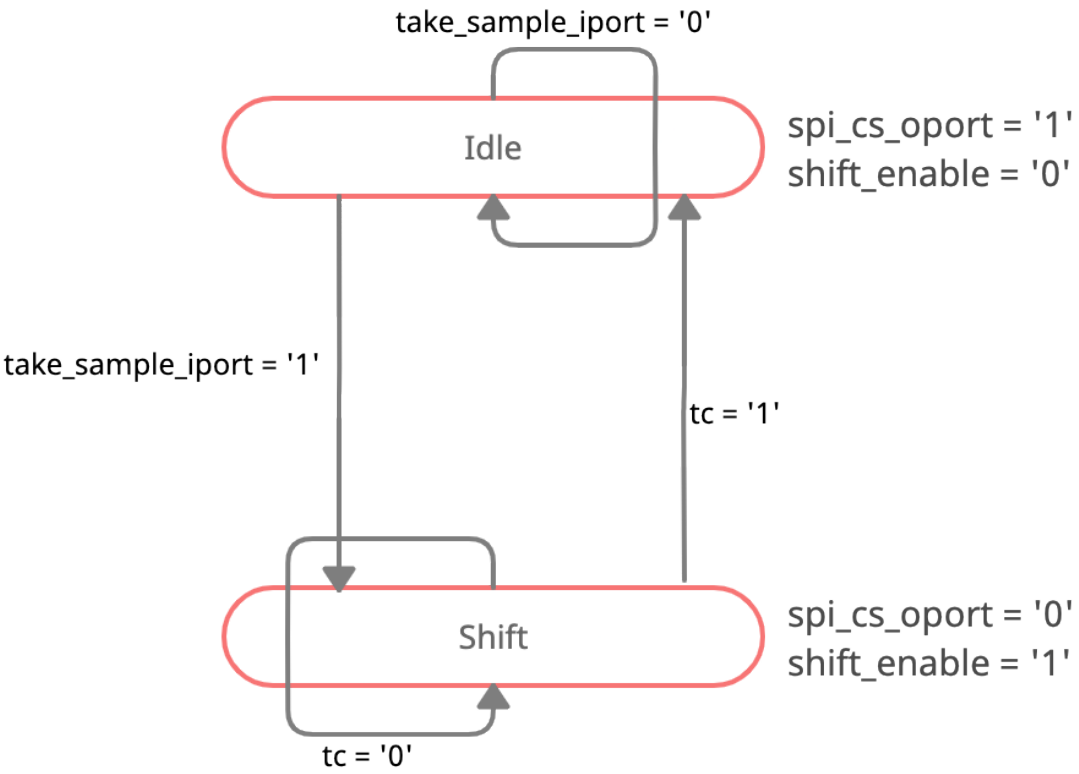
1.3: Parts List

- Pmod DA2
- Pmod DA2 to AMP2 adapter
- Pmod AMP2
- MIDI to Pmod adapter
- Alesis Q25 MIDI keyboard
- MIDI keyboard USB power in
- Male to Male 5-pin Midi Cable

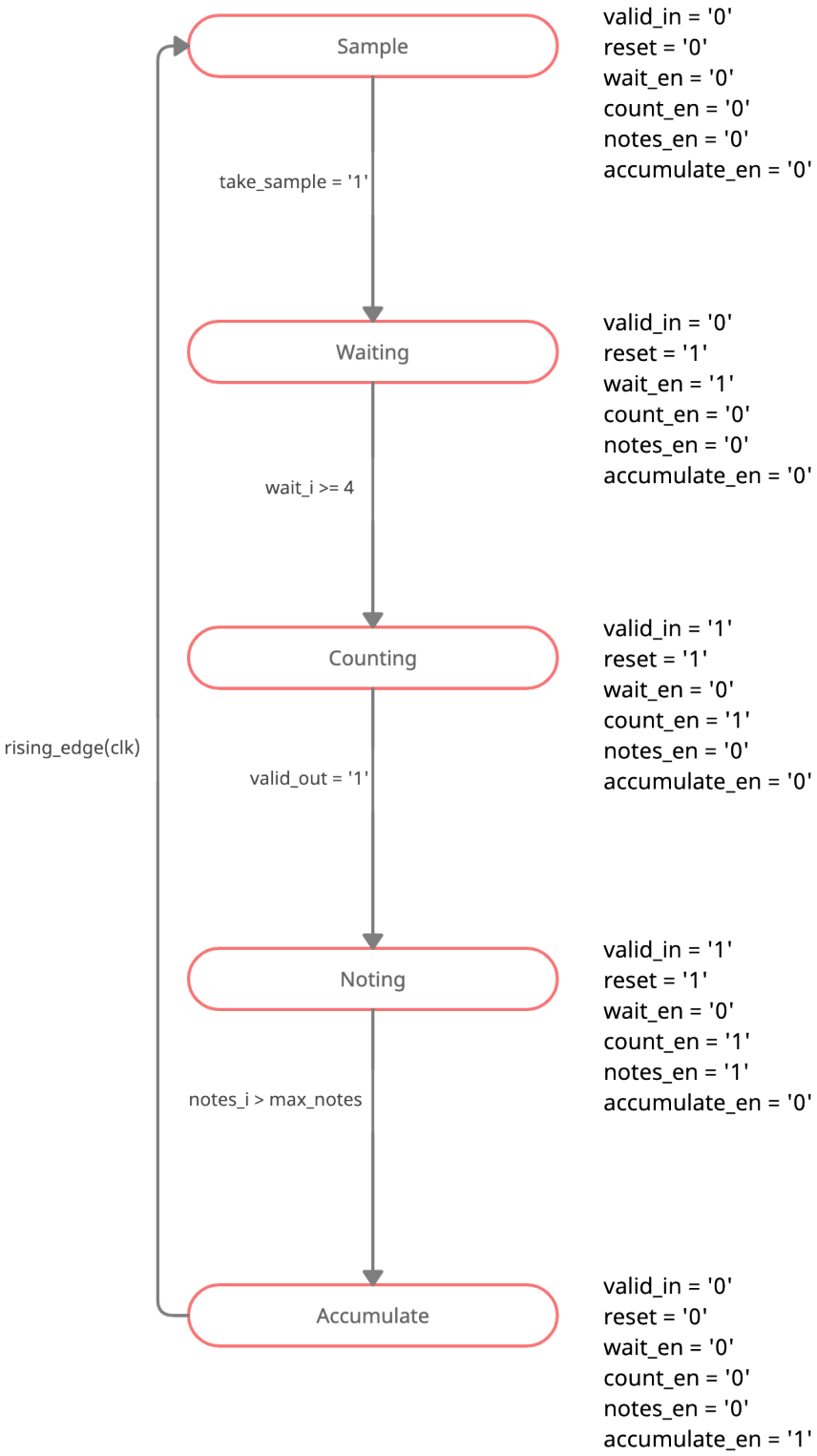
Appendix 2: Programmed Logic

2.1: State Diagrams

2.1.1: Digital-to-analog converter shift register

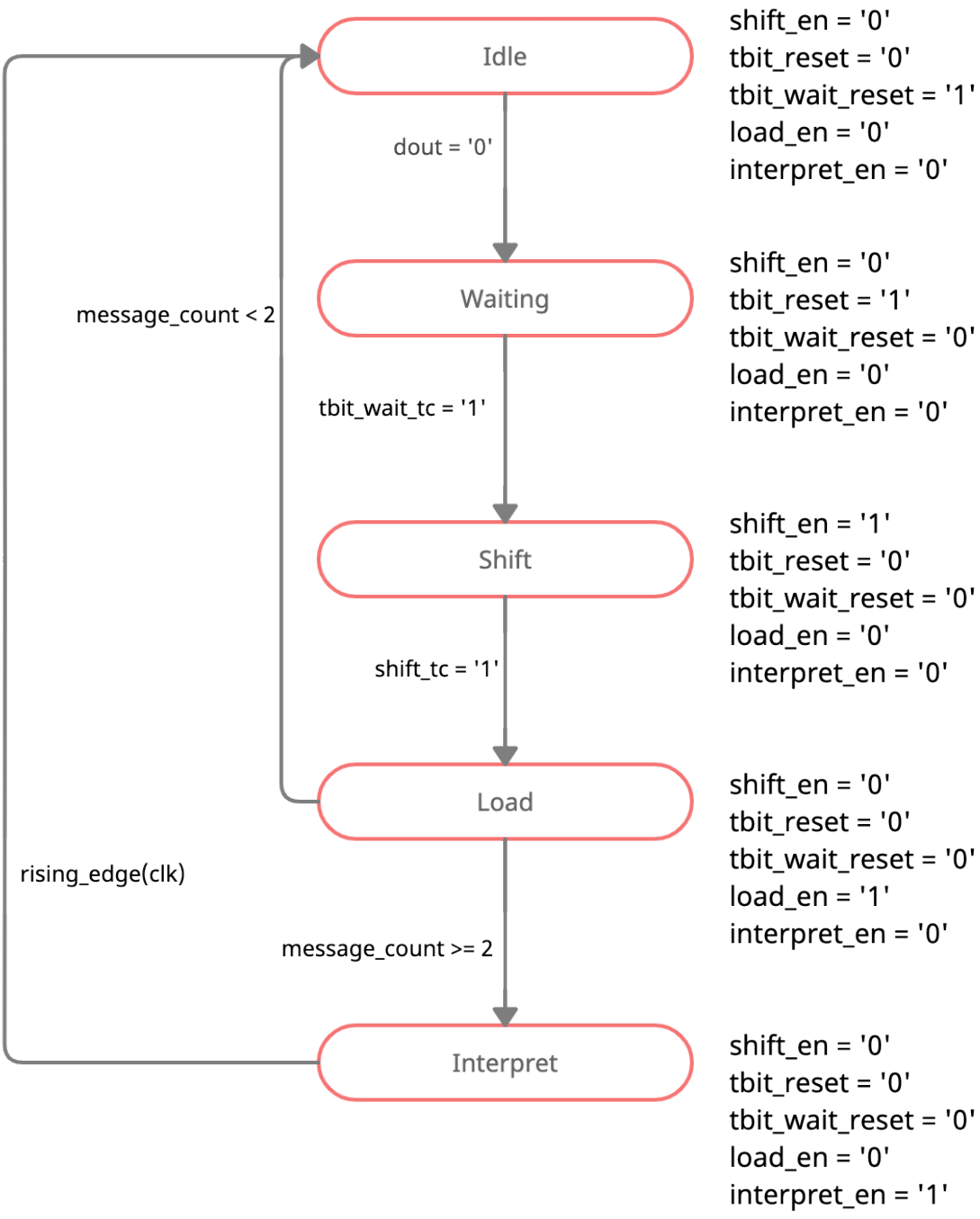


2.1.2: Note synthesizer



Note that wait_i and notes_i are ongoing counts, represented internally by signals and activated by the wait_en and count_en flags. valid_out is provided by the sine wave LUT.

2.1.3: MIDI keyboard interface



Note that dout refers to the signal which is the output of the double-flop synchronizer. tbit_wait_tc and shift_tc are the result of a tick generator that outputs a tick every tbit/2 µs and the terminal count of the shift counter, respectively. message_count is incremented in the load state.

2.2: VHDL Code

2.2.1: project_shell.vhd

```
----------------------------------------------------------------------------
------------
-- Company: ENGS 31 / COSC 56
-- Engineer: Tyler Vergho, Arnav Tolat
--
-- Create Date: 08/17/2021 03:19:18 PM
-- Design Name:
-- Module Name: project_shell - Behavioral
-- Project Name: Group 5 MIDI Synthesizer
-- Target Devices: Basys 3
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
--
----------------------------------------------------------------------------
------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;

--================================================================
--Shell Entity Declarations
--================================================================

entity project_shell is
port (
    clk_iport_100MHz  : in  std_logic;
    midi_iport : in std_logic;
    spi_cs_oport      : out std_logic;
    spi_sclk_oport          : out std_logic;
    spi_data_oport : out std_logic;
```

```
    seg_oport              : out std_logic_vector(0 to 6);
--segment control
     dp_oport              : out std_logic;
--decimal point control
     an_oport              : out std_logic_vector(3 downto 0)
--digit control
);
end project_shell;

--================================================================
--Architecture Declarations
--================================================================
architecture Behavioral of project_shell is
component clk_wiz_0 is
  Port (
    clk_out1 : out STD_LOGIC;
    reset : in STD_LOGIC;
    locked : out STD_LOGIC;
    clk_in1 : in STD_LOGIC
  );
end component;

component midi_keyboard is
    Generic (max_notes : integer);
    Port ( data_iport : in STD_LOGIC;
           clk_24mHz : in STD_LOGIC;
           notes_oport : out STD_LOGIC_VECTOR(0 to max_notes) :=
(others => '0'));
end component;

component tick_generator is
     generic (
        FREQUENCY_DIVIDER_RATIO : integer);
     port (
          system_clk_iport : in  std_logic;
          tick_oport       : out std_logic);
end component;

component d_to_a is
     generic(
          N_SHIFTS             : integer);
```

```
     port(
--that can be treated as constants in this level.
          clk_iport                  : in  std_logic; --1 MHz serial
clock
     dac_data_iport             : in std_logic_vector(15 downto 0);

          take_sample_iport          : in  std_logic; --controller
signals
          spi_cs_oport               : out std_logic;

          dac_data_oport             : out std_logic);
end component;

component synthesizer is
    Generic ( max_notes : integer);
    Port ( take_sample : in STD_LOGIC;
           notes_on: in STD_LOGIC_VECTOR(0 to max_notes);
           note_out : out STD_LOGIC_VECTOR(15 DOWNTO 0);
           clk_24mHz : in STD_LOGIC
           );
end component;

component mux7seg is
    Port ( clk_iport  : in  std_logic;                              --
runs on a fast (1 MHz or so) clock
          y3_iport    : in  std_logic_vector (3 downto 0);  -- digits
          y2_iport    : in  std_logic_vector (3 downto 0);  -- digits
      y1_iport        : in  std_logic_vector (3 downto 0);  -- digits
          y0_iport    : in  std_logic_vector (3 downto 0);  -- digits
       dp_set_iport : in  std_logic_vector(3 downto 0);     -- decimal
points
          seg_oport  : out std_logic_vector(0 to 6);             --
segments (a...g)
          dp_oport   : out std_logic;                            --
decimal point
          an_oport   : out std_logic_vector (3 downto 0) );-- anodes
end component;

--================================================================
--Local Signal Declarations
--================================================================
constant max_notes : integer := 87;
```

```
signal clk_24mHz : std_logic := '0';
signal take_sample : std_logic := '0';
signal note : std_logic_vector(15 downto 0) := (others => '0'); --
synthesizer output, overall note being played
signal locked : std_logic := '0'; -- timing
signal switches_actual, switches_filtered, switches_old :
std_logic_vector(0 to max_notes) := (others => '0');

signal y3: std_logic_vector(3 downto 0) := (others => '0');
signal y2: std_logic_vector(3 downto 0) := (others => '0');
signal y1: std_logic_vector(3 downto 0) := (others => '0');
signal y0: std_logic_vector(3 downto 0) := (others => '0');
signal note_out_code : std_logic_vector(15 downto 0) := (others =>
'0');

begin

--================================================================
-- Port Mapping
--================================================================
clocking: clk_wiz_0
port map(
      clk_in1 => clk_iport_100MHz,
      reset => '0',
      clk_out1 => clk_24mHz,
      locked => locked
);

tick_generation: tick_generator
generic map(
      FREQUENCY_DIVIDER_RATIO => 500)
port map(
      system_clk_iport      => clk_24mHz,
      tick_oport            => take_sample);

converter: d_to_a
generic map ( N_SHIFTS => 16 )
port map (
    clk_iport => clk_24mHz,
    dac_data_iport => note,
    take_sample_iport => take_sample,
    spi_cs_oport => spi_cs_oport,
```

```
    dac_data_oport => spi_data_oport
);


spi_sclk_oport <= clk_24mHz;

synth: synthesizer
generic map (max_notes => max_notes)
port map (
    take_sample => take_sample,
    notes_on => switches_filtered,
    note_out => note,
    clk_24mHz => clk_24mHz
);

midi: midi_keyboard
generic map (max_notes => max_notes)
port map (
    data_iport => midi_iport,
    notes_oport => switches_actual,
    clk_24mHz => clk_24mHz
);

note_off_clock: process(clk_24mHz)
begin
    if rising_edge(clk_24mHz) then
        switches_old <= switches_filtered;
    end if;
end process note_off_clock;

-- only stop playing the notes when the overall amplitude < 128
-- gets rid of the clicking sound when the note terminates
note_off_filter: process(switches_actual, note, switches_old)
begin
    switches_filtered <= switches_old;

    if unsigned(switches_actual) = 0 then
        if unsigned(note(15 downto 8)) = 0 then
            switches_filtered <= switches_actual;
        end if;
    else
        switches_filtered <= switches_actual;
```

```
    end if;
end process note_off_filter;

-- correlates keys being pressed with the code on the display
set_note_out: process (switches_actual)
begin
    note_out_code <= (others => '0');
    note_loop: for i in 0 to max_notes loop -- go through all the
notes
        if switches_actual(i) = '1' then
            note_out_code <= std_logic_vector(to_unsigned(i + 21,
16)); -- set the code to the highest depressed note
        end if;
   end loop note_loop;
end process set_note_out;

set_digits: process (note_out_code)
begin
    y3 <= note_out_code(15 downto 12);
    y2 <= note_out_code(11 downto 8);
    y1 <= note_out_code(7 downto 4);
    y0 <= note_out_code(3 downto 0);
end process set_digits;

display: mux7seg port map(
    clk_iport         => clk_24mHz,
    y3_iport          => y3,
    y2_iport          => y2,
    y1_iport          => y1,
    y0_iport          => y0,
    dp_set_iport => "0000",
    seg_oport         => seg_oport,
    dp_oport          => dp_oport,
    an_oport          => an_oport );

end Behavioral;
```

2.2.2: d_a_converter.vhd

```
--------------------------------------------------------------
--Ben Dobbins
--ES31/CS56
--Final
--Your name goes here: Arnav Tolat & Tyler Vergho
--------------------------------------------------------------

--------------------------------------------------------------
--Library Declarations
--------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;             -- needed for arithmetic
use ieee.math_real.all;                    -- needed for automatic
register sizing
library UNISIM;                            -- needed for the BUFG
component
use UNISIM.Vcomponents.ALL;

--------------------------------------------------------------
--Shell Entity Declarations
--------------------------------------------------------------
entity d_to_a is
     generic(
          N_SHIFTS                  : integer);
port( clk_iport                  : in  std_logic; --24 MHz serial clock
     dac_data_iport              : in std_logic_vector(15 downto 0);
     take_sample_iport           : in  std_logic; --controller signals
     spi_cs_oport                : out std_logic;
     dac_data_oport              : out std_logic);
end d_to_a;

--------------------------------------------------------------
--Architecture Declaration
--------------------------------------------------------------
architecture Behavioral of d_to_a is
--------------------------------------------------------------
--Local Signal Declaration
--------------------------------------------------------------
--Your controller signal declarations go here:
```

```
signal count: unsigned(5 downto 0) := (others => '0');
signal tc: std_logic := '0';
signal shift_enable: std_logic := '0';

type state_type is (idle, shift);
signal curr_state, next_state: state_type;
signal filtered : std_logic_vector(15 downto 0) := (others => '0');

begin
--================================================================
--Controller:
--================================================================
--Your controller goes here
timer: process(clk_iport, count)
begin
     if rising_edge(clk_iport) then
          if (shift_enable = '1') then
               count <= count+1;
          else
              count <= (others => '0');
          end if;

          if tc = '1' then
            count <= (others => '0');
          end if;
     end if;

     if (count >= N_SHIFTS) then
        tc <= '1';
    else
        tc <= '0';
    end if;
end process timer;

FSM_comb: process(curr_state, tc, take_sample_iport)
begin
    next_state <= curr_state;
    spi_cs_oport <= '1';
    shift_enable <= '0';

    -- states and transitions, as reflected in state diagram
```

```
    case curr_state is
        when idle => -- idle starting state, which remains until take
sample is high
            if take_sample_iport = '1' then -- when take sample is
high transition to shift state
                shift_enable <= '1';
                next_state <= shift;
            end if;

        when shift => -- shifting state, remains until timeout occurs
            shift_enable <= '1';
            spi_cs_oport <= '0'; --goes low for shift, but returns
high for other states since its default value is 1
            if tc = '1' then
                next_state <= idle; -- transition to idle after TC
            end if;
    end case;
end process FSM_comb;

FSM_update: process(clk_iport) -- clocked update process
begin
    if rising_edge(clk_iport) then
        curr_state <= next_state; -- rising edge of clock, current
state updates
    end if;
end process FSM_update;
--=============================================================
--Datapath:
--=============================================================
shift_register: process(clk_iport)
begin
     if rising_edge(clk_iport) then
        if to_integer(N_SHIFTS-1-count) >= 0 and
to_integer(N_SHIFTS-1-count) <= 15 then
            dac_data_oport <= filtered(to_integer(N_SHIFTS-1-count));
        else
            dac_data_oport <= '0';
        end if;
    end if;
end process shift_register;
```

```
-- right shift by 1 bit to divide the amplitude in half
filter: process (dac_data_iport)
begin
    filtered <= '0' & dac_data_iport(15 downto 1);
end process filter;

end Behavioral;
```

2.2.3: synthesizer.vhd

```
----------------------------------------------------------------------------
------------
-- Company: ENGS 31 / COSC 56
-- Engineer: Tyler Vergho, Arnav Tolat
--
-- Create Date: 08/18/2021 03:47:41 PM
-- Design Name: Synthesizer
-- Module Name: synthesizer - Behavioral
-- Project Name: Group 5 MIDI Synthesizer
-- Target Devices: Basys 3
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
--
----------------------------------------------------------------------------
------------

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

--================================================================
--Shell Entity Declarations
--================================================================

entity synthesizer is
    Generic ( max_notes : integer);
    Port ( take_sample : in STD_LOGIC;
           notes_on: in STD_LOGIC_VECTOR(0 to max_notes);
           note_out : out STD_LOGIC_VECTOR(15 DOWNTO 0);
           clk_24mHz : in STD_LOGIC);
end synthesizer;

--================================================================
--Architecture Declarations
--================================================================
```

```
architecture Behavioral of synthesizer is
component dds is
    Generic (STEP_SIZE : integer);
    Port ( clk_iport : in STD_LOGIC;
           count_en : in STD_LOGIC;
           count_oport : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) );
end component;

COMPONENT dds_compiler_0
  PORT (
    aclk : IN STD_LOGIC;
    aresetn: IN STD_LOGIC;
    s_axis_phase_tvalid : IN STD_LOGIC;
    s_axis_phase_tdata : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
    m_axis_data_tvalid : OUT STD_LOGIC;
    m_axis_data_tdata : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
  );
END COMPONENT;

type steps_t is array (0 to max_notes) of integer;
type notes_t is array (0 to max_notes) of STD_LOGIC_VECTOR(15 DOWNTO
0);
-- LUT of accumulator step sizes in memory
constant steps : steps_t := (
    0 => 19, -- 27.5
    1 => 20, -- 29.135
    2 => 21, -- 30.868
    3 => 22, -- 32.703
    4 => 24, -- 34.648
    5 => 25,  -- 36.708
    6 => 27, -- 38.891
    7 => 28, -- 41.203
    8 => 30, -- 43.654
    9 => 32, -- 46.249
    10 => 33, -- 48.999
    11 => 35, -- 51.913
    12 => 38, -- 55.000
    13 => 40, -- 58.270
    14 => 42, -- 61.735
    15 => 45, -- 65.406
    16 => 47, -- 69.296
    17 => 50, -- 73.416
```

```
18 => 53, -- 77.782
19 => 56, -- 82.407
20 => 60, -- 87.307
21 => 63, -- 92.499
22 => 67, -- 97.999
23 => 71, -- 103.83
24 => 75, -- 110
25 => 80, -- 116.54
26 => 84, --123.47
27 => 89, -- 130.81
28 => 95, -- 138.59
29 => 100, -- 146.83
30 => 106, -- 155.56
31 => 113, -- 164.81
32 => 119, -- 174.61
33 => 126, -- 185
34 => 134, -- 196
35 => 142, -- 207.65
36 => 150, -- 220
37 => 159, -- 233.08
38 => 169, -- 246.94
39 => 179, -- 261.63
40 => 189, -- 277.18
41 => 201, -- 293.67
42 => 212, -- 311.13
43 => 225, -- 329.63
44 => 238, -- 349.23
45 => 253, -- 369.99
46 => 268, -- 392.00
47 => 284, -- 415.30
48 => 300, -- 440.00
49 => 318, -- 466.16
50 => 337, -- 493.88
51 => 357, -- 523.25
52 => 378, -- 554.37
53 => 401, -- 587.33
54 => 425, -- 622.25
55 => 450, -- 659.26
56 => 477, -- 698.46
57 => 505, -- 739.99
58 => 535, -- 783.99
59 => 567, -- 830.61
```

```
    60 => 601, -- 880.00
    61 => 636, -- 932.33
    62 => 674, -- 987.77
    63 => 714, -- 1046.5
    64 => 756, -- 1108.7
    65 => 802, -- 1174.7
    66 => 850, -- 1244.5
    67 => 900, --1318.5
    68 => 954, -- 1396.9
    69 => 1010, -- 1480
    70 => 1070, -- 1568
    71 => 1134, -- 1661.2
    72 => 1201, -- 1760
    73 => 1273, -- 1864.7
    74 => 1349, -- 1975.5
    75 => 1429, -- 2093
    76 => 1514, -- 2217.5
    77 => 1604, -- 2349.3
    78 => 1699, -- 2489
    79 => 1800, -- 2637
    80 => 1907,  -- 2793
    81 => 2021, -- 2960
    82 => 2141, -- 3136
    83 => 2268, -- 3322.4
    84 => 2403, --3520
    85 => 2546, -- 3729.3
    86 => 2697, -- 3951.1
    87 => 2858 -- 4186
);
signal counts, notes : notes_t := (others => (others => '0'));
signal count, note : STD_LOGIC_VECTOR(15 DOWNTO 0) := (others => '0');
signal data_in, data_out : std_logic_vector(15 downto 0) := (others =>
'0');
signal valid_in, valid_out : std_logic := '0';
signal reset : std_logic := '1';

signal count_en, notes_en, wait_en, accumulate_en : std_logic := '0';
signal count_i, notes_i, wait_i : integer := 0;

type state_type is (SAMPLE, WAITING, COUNTING, NOTING, ACCUMULATE);
signal curr_state, next_state: state_type;
```

```
signal numnotes_sig : unsigned(5 downto 0) := (others => '0');
signal notes_accumulated_sig : unsigned(17 downto 0) := (others =>
'0');

begin
gen_sine: -- one phase accumulator for each possible note
for I in 0 to max_notes generate
    sine_I: dds
    generic map ( STEP_SIZE => steps(I) )
    port map ( clk_iport => take_sample,
           count_en => notes_on(i),
           count_oport => counts(i)
    );
end generate gen_sine;

-- convert sine LUT two's complement output to binary offset
count_to_sin: process (count, data_out)
begin
    data_in <= std_logic_vector('0' & count(14 downto 0));
    note <= "0000" & not(data_out(11)) & data_out(10 downto 0);
end process count_to_sin;

sin_lut : dds_compiler_0
  PORT MAP (
    aclk => clk_24mHz,
    aresetn => reset,
    s_axis_phase_tvalid => valid_in,
    s_axis_phase_tdata => data_in,
    m_axis_data_tvalid => valid_out,
    m_axis_data_tdata => data_out
);

--================================================================
--Controller
--================================================================
FSMupdate: process(clk_24mHz)
begin
    if rising_edge(clk_24mHz) then
       curr_state <= next_state;
    end if;
end process FSMupdate;
```

```
FSM_count: process (clk_24mHz)
begin
    if rising_edge(clk_24mHz) then
        if notes_en = '1' then
            if notes_i <= max_notes then
                notes(notes_i) <= note;
            end if;
            notes_i <= notes_i + 1;
        else
            notes_i <= 0;
        end if;

        if count_en = '1' then
            if count_i <= max_notes then
                count <= counts(count_i);
            end if;
            count_i <= count_i + 1;
        else
            count_i <= 0;
        end if;

        if wait_en = '1' then
            wait_i <= wait_i + 1;
        else
            wait_i <= 0;
        end if;
    end if;
end process FSM_count;

accumulator: process (curr_state, wait_i, notes, count_i, valid_out,
notes_i, take_sample, counts, note)
begin
    next_state <= curr_state;
    valid_in <= '0';
    reset <= '0';
    wait_en <= '0';
    count_en <= '0';
    notes_en <= '0';
    accumulate_en <= '0';

    case curr_state is
        when SAMPLE =>
```

```
            if take_sample = '1' then
                next_state <= WAITING;
            end if;
        when WAITING => reset <= '1';
            wait_en <= '1';
            if wait_i >= 4 then
                next_state <= COUNTING;
            end if;
        when COUNTING => valid_in <= '1';
            reset <= '1';
            count_en <= '1';

            if valid_out = '1' then
                next_state <= NOTING;
            end if;
         when NOTING => valid_in <= '1';
            reset <= '1';
            notes_en <= '1';
            count_en <= '1';

            if notes_i > max_notes then
                next_state <= ACCUMULATE;
            end if;

          when ACCUMULATE => accumulate_en <= '1';
                next_state <= SAMPLE;
    end case;
end process accumulator;

-- loop through each of the notes during the accumulate state
-- and update the notes_accumulated and numnotes signals
proc_notes: process(clk_24mHz)
variable notes_accumulated : unsigned(17 downto 0) := (others => '0');
variable numnotes : unsigned(5 downto 0) := (others => '0');

begin
    if rising_edge(clk_24mHz) then
        if accumulate_en = '1' then
            notes_accumulated := (others => '0');
            numnotes := (others => '0');
            note_loop: for i in 0 to max_notes loop
                if notes_on(i) = '1' then
```

```
                    notes_accumulated := notes_accumulated +
unsigned(notes(i));
                    numnotes := numnotes + 1;
                end if;
            end loop note_loop;

            numnotes_sig <= numnotes;
            notes_accumulated_sig <= notes_accumulated;
        end if;
    end if;
end process proc_notes;

-- unclocked arithmetic process to avoid excessively long propagation
delay
arithmetic: process(numnotes_sig, notes_accumulated_sig)
begin
    note_out <= (others => '0');
    if numnotes_sig > 0 then
        -- divide by the number of notes to allow polyphony
        note_out <=
STD_LOGIC_VECTOR(resize(notes_accumulated_sig/numnotes_sig, 16));
    end if;
end process arithmetic;

end Behavioral;
```

2.2.4: midi_keyboard.vhd

```
----------------------------------------------------------------------------
------------
-- Company: ENGS 31 / COSC 56
-- Engineer: Tyler Vergho, Arnav Tolat
--
-- Create Date: 08/20/2021 07:18:59 PM
-- Design Name: MIDI Keyboard Interface
-- Module Name: midi_keyboard - Behavioral
-- Project Name: Group 5 MIDI Synthesizer
-- Target Devices: Basys 3
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
--
----------------------------------------------------------------------------
------------

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

--================================================================
--Shell Entity Declarations
--================================================================
entity midi_keyboard is
    Generic (max_notes : integer);
    Port ( data_iport : in STD_LOGIC;
           clk_24mHz : in STD_LOGIC;
           notes_oport : out STD_LOGIC_VECTOR(0 to max_notes) :=
(others => '0'));
end midi_keyboard;

--================================================================
--Architecture Declarations
--================================================================
architecture Behavioral of midi_keyboard is
```

```
--============================================================
--Local Signals and Constants
--============================================================
constant TBIT_MAX : integer := 768; -- based on a 24 mHz clock
constant N_SHIFTS : integer := 10;
constant NOTE_ON : STD_LOGIC_VECTOR(3 downto 0) := "1001";
constant NOTE_OFF : STD_LOGIC_VECTOR(3 downto 0) := "1000";
type state_type is (IDLE, WAITING, LOAD, SHIFT, INTERPRET);
signal curr_state, next_state : state_type;

signal shift_en, load_en, interpret_en, shift_tc, tbit_tc, tbit_reset,
tbit_wait_tc, tbit_wait_reset : STD_LOGIC := '0';
signal tbit_count : unsigned(7 downto 0) := (others => '0');
signal shift_count : unsigned(4 downto 0) := (others => '0');

type messages_t is array(0 to 2) of std_logic_vector(7 downto 0);
signal messages : messages_t := (others => (others => '0'));
signal cur_message : std_logic_vector(9 downto 0) := (others => '0');
signal message_count : unsigned(1 downto 0) := (others => '0');
signal dsync, dout : std_logic := '1';

--============================================================
--Component Declarations
--============================================================
component tick_generator is
     generic (
        FREQUENCY_DIVIDER_RATIO : integer);
     port (
          system_clk_iport : in  std_logic;
          reset_iport : in std_logic;
          tick_oport       : out std_logic);
end component;

begin
sync: process (clk_24mHz) -- D flip-flop synchronizer
begin
    if rising_edge(clk_24mHz) then
        dsync <= data_iport;
        dout <= dsync;
    end if;
end process sync;
```

```
-- counter to support MIDI timing (31,250 bits/sec)
shift_counter: process(clk_24mHz, shift_count)
begin
    if rising_edge(clk_24mHz) then
        if tbit_tc = '1' then
            if shift_en = '1' then
                shift_count <= shift_count + 1;
            end if;
        end if;
        if shift_en = '0' then
            shift_count <= (others => '0');
        end if;
    end if;

    shift_tc <= '0';
    if shift_count >= N_SHIFTS - 1 then
        shift_tc <= '1';
    end if;
end process shift_counter;


FSMupdate: process(clk_24mHz)
begin
    if rising_edge(clk_24mHz) then
       curr_state <= next_state;
    end if;
end process FSMupdate;

-- on the INTERPRET state, updates the notes_out register based on the
message received
set_notes: process(clk_24mHz)
begin
    if rising_edge(clk_24mHz) then
        if interpret_en = '1' then
            if messages(0)(7 downto 4) = NOTE_ON then
                if (unsigned(messages(1)) - 21) <= max_notes then
                    notes_oport(to_integer(unsigned(messages(1)) -
21)) <= '1';
                end if;
            elsif messages(0)(7 downto 4) = NOTE_OFF then
                if (unsigned(messages(1)) - 21) <= max_notes then
```

```
                    notes_oport(to_integer(unsigned(messages(1)) -
21)) <= '0';
                end if;
            end if;
        end if;
    end if;
end process set_notes;

--=============================================================
--Controller
--=============================================================
FSMcomb: process(curr_state, dout, tbit_tc, tbit_wait_tc, shift_tc,
message_count)
begin
    next_state <= curr_state;
    shift_en <= '0';
    tbit_reset <= '0';
    tbit_wait_reset <= '0';
    load_en <= '0';
    interpret_en <= '0';

    case curr_state is
        when IDLE =>
            if dout = '0' then
                tbit_wait_reset <= '1';
                next_state <= WAITING;
            end if;
       when WAITING => tbit_reset <= '1';
            if tbit_wait_tc = '1' then
                next_state <= SHIFT;
            end if;
       when SHIFT => shift_en <= '1';
            if shift_tc = '1' then
                next_state <= LOAD;
            end if;
       when LOAD => load_en <= '1';
            if message_count < 2 then
                next_state <= IDLE;
            else
                next_state <= INTERPRET;
            end if;
       when INTERPRET => interpret_en <= '1';
```

```
            next_state <= IDLE;
    end case;
end process FSMcomb;

shift_register: process(clk_24mHz)
begin
    if rising_edge(clk_24mHz) then
         if tbit_tc = '1' then
              if shift_en = '1' then cur_message <= dout & cur_message(9
downto 1);
              end if;
        end if;
    end if;
end process;

shift_load: process(clk_24mHz)
begin
    if rising_edge(clk_24mHz) then
        if load_en = '1' then
             messages(to_integer(message_count)) <= cur_message(8
downto 1);
             if message_count < 2 then
            message_count <= message_count + 1;
          else
            message_count <= (others => '0');
          end if;
           end if;
    end if;
end process shift_load;

-- 32 us tick
tick_generation: tick_generator
generic map(
     FREQUENCY_DIVIDER_RATIO => TBIT_MAX)
port map(
     system_clk_iport      => clk_24mHz,
     reset_iport => tbit_reset,
     tick_oport            => tbit_tc);

-- 16 us tick
tick_generation_2: tick_generator
generic map(
```

```
      FREQUENCY_DIVIDER_RATIO => TBIT_MAX/2)
port map(
      system_clk_iport      => clk_24mHz,
      reset_iport => tbit_wait_reset,
      tick_oport            => tbit_wait_tc);

end Behavioral;
```

2.2.5: dds.vhd

```
----------------------------------------------------------------------------
------------
-- Company: ENGS 31 / COSC 56
-- Engineer: Tyler Vergho, Arnav Tolat
--
-- Create Date: 08/15/2021 08:04:10 PM
-- Design Name: DDS Phase Accumulator
-- Module Name: dds - Behavioral
-- Project Name: Group 5 MIDI Synthesizer
-- Target Devices: Basys 3
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
--
----------------------------------------------------------------------------
------------

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

--================================================================
--Shell Entity Declarations
--================================================================
entity dds is
    Generic (STEP_SIZE : integer);
    Port ( clk_iport : in STD_LOGIC;
           count_en : in STD_LOGIC;
           count_oport : OUT STD_LOGIC_VECTOR(15 DOWNTO 0) );
end dds;

--================================================================
--Architecture Declarations
--================================================================
architecture Behavioral of dds is
```

```
signal count : unsigned(14 downto 0) := (others => '0'); -- 15-bit
address

begin
increment: process(clk_iport, count)
begin
    if rising_edge(clk_iport) then
        if count_en = '1' then
            count <= count + STEP_SIZE;
        else
            count <= "110000000000000"; -- start at 24576 (bottom of
sine wave)
        end if;
    end if;
    count_oport <= '0' & std_logic_vector(count);
end process increment;

end Behavioral;
```

2.2.6: tick_generator.vhd

```
--================================================================================
=========
--Library Declarations:
--================================================================================
=========
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.math_real.all;
library UNISIM;
use UNISIM.VComponents.all;

--================================================================================
=========
--Entity Declaration:
--================================================================================
=========
entity tick_generator is
      generic (FREQUENCY_DIVIDER_RATIO : integer);
      port (
            system_clk_iport : in  std_logic;
            reset_iport : in std_logic := '0';
            tick_oport        : out std_logic);
end tick_generator;

--================================================================================
=========
--Architecture Type:
--================================================================================
=========
architecture behavioral_architecture of tick_generator is
--================================================================================
=========
--Signal Declarations:
--================================================================================
=========
--CONSTANT FOR SYNTHESIS:
constant FREQUENCY_DIVIDER_TC: integer := FREQUENCY_DIVIDER_RATIO;
--CONSTANT FOR SIMULATION:
--constant FREQUENCY_DIVIDER_TC: integer := 20;
```

```
--Automatic register sizing:
constant COUNT_LEN                          : integer := integer(ceil(
log2( real(FREQUENCY_DIVIDER_TC) ) ));
signal frequency_divider_counter : unsigned(COUNT_LEN-1 downto 0) :=
(others => '0');

--=================================================================
=========
--Processes:
--=================================================================
=========
begin
--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++
--Frequency Divider:
--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++
frequency_divider: process(system_clk_iport,
frequency_divider_counter)
begin
      if rising_edge(system_clk_iport) then
            if frequency_divider_counter = FREQUENCY_DIVIDER_TC-1 or
reset_iport = '1' then
                  frequency_divider_counter <= (others => '0');
        -- Reset
            else
                  frequency_divider_counter <= frequency_divider_counter
+ 1; -- Count up
            end if;
      end if;

      if frequency_divider_counter = FREQUENCY_DIVIDER_TC-1 then
tick_oport <= '1';
      else tick_oport <= '0';
      end if;
end process frequency_divider;


end behavioral_architecture;
```

2.2.7: mux7seg.vhd

```
----------------------------------------------------------------------
------------
-- Company:                 Engs 31 16X
-- Engineer:                E.W. Hansen
--
-- Create Date:           17:56:35 07/25/2008
-- Design Name:
-- Module Name:           mux7seg - Behavioral
-- Project Name:
-- Target Devices:        Digilent Basys 3 board (Artix 7)
-- Tool versions:         Vivado 2016.1
-- Description:       Multiplexed seven-segment decoder for the display
on the Basys3
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Revision 1.00 (07/17/2015) --- drop the clock divider, run on a
1000 Hz clock
-- Revision 2.00 (07/17/2016) --- put the clock divider back in
-- Additional Comments:
--
----------------------------------------------------------------------
------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity mux7seg is
    Port ( clk_iport  : in  std_logic;                              --
runs on a fast (1 MHz or so) clock
            y3_iport  : in  std_logic_vector (3 downto 0);  -- digits
             y2_iport     : in  std_logic_vector (3 downto 0);  --
digits
             y1_iport     : in  std_logic_vector (3 downto 0);  --
digits
          y0_iport   : in  std_logic_vector (3 downto 0);  -- digits
          dp_set_iport : in  std_logic_vector(3 downto 0);     --
decimal points
```

```
           seg_oport  : out  std_logic_vector(0 to 6);      --
segments (a...g)
           dp_oport   : out  std_logic;
-- decimal point
           an_oport   : out  std_logic_vector (3 downto 0) );     --
anodes
end mux7seg;

architecture Behavioral of mux7seg is
     constant NCLKDIV:     integer := 11;                      -- 1 MHz
/ 2^18 = 381 Hz
    constant MAXCLKDIV:   integer := 2**NCLKDIV-1;           -- max
count of clock divider
    signal cdcount:       unsigned(NCLKDIV-1 downto 0);      -- clock
divider counter register
    signal CE :           std_logic;                         -- clock
enable

     signal adcount : unsigned(1 downto 0) := "00";              --
anode / mux selector count
     signal anb: std_logic_vector(3 downto 0);
     signal muxy : std_logic_vector(3 downto 0);                 --
mux output
     signal segh : std_logic_vector(0 to 6);
-- segments (high true)

begin
-- Clock divider sets the rate at which the display hops from one
digit to the next.  A larger value of
-- MAXCLKDIV results in a slower clock-enable (CE)
ClockDivider:
process(clk_iport)
begin
     if rising_edge(clk_iport) then
        if cdcount < MAXCLKDIV then
             CE <= '0';
                cdcount <= cdcount+1;
        else
                CE <= '1';
             cdcount <= (others => '0');
        end if;
     end if;
```

```
end process ClockDivider;

AnodeDriver:
process(clk_iport, adcount)
begin
      if rising_edge(clk_iport) then
         if CE='1' then
              adcount <= adcount + 1;
         end if;
      end if;

      case adcount is
            when "00" => anb <= "1110";
            when "01" => anb <= "1101";
            when "10" => anb <= "1011";
            when "11" => anb <= "0111";
            when others => anb <= "1111";
      end case;
end process AnodeDriver;

an_oport <= anb;

Multiplexer:
process(adcount, y0_iport, y1_iport, y2_iport, y3_iport, dp_set_iport)
begin
      case adcount is
            when "00" => muxy <= y0_iport; dp_oport <=
not(dp_set_iport(0));
            when "01" => muxy <= y1_iport; dp_oport <=
not(dp_set_iport(1));
            when "10" => muxy <= y2_iport; dp_oport <=
not(dp_set_iport(2));
            when "11" => muxy <= y3_iport; dp_oport <=
not(dp_set_iport(3));
            when others => muxy <= x"0"; dp_oport <= '1';
      end case;
end process Multiplexer;

-- Seven segment decoder
with muxy select segh <=
      "1111110" when x"0",        -- active-high definitions
      "0110000" when x"1",
```

```
      "1101101" when x"2",
      "1111001" when x"3",
      "0110011" when x"4",
      "1011011" when x"5",
      "1011111" when x"6",
      "1110000" when x"7",
      "1111111" when x"8",
      "1111011" when x"9",
      "1110111" when x"a",
      "0011111" when x"b",
      "1001110" when x"c",
      "0111101" when x"d",
      "1001111" when x"e",
      "1000111" when x"f",
      "0000000" when others;
seg_oport <= not(segh);                    -- Convert to active-low

end Behavioral;
```

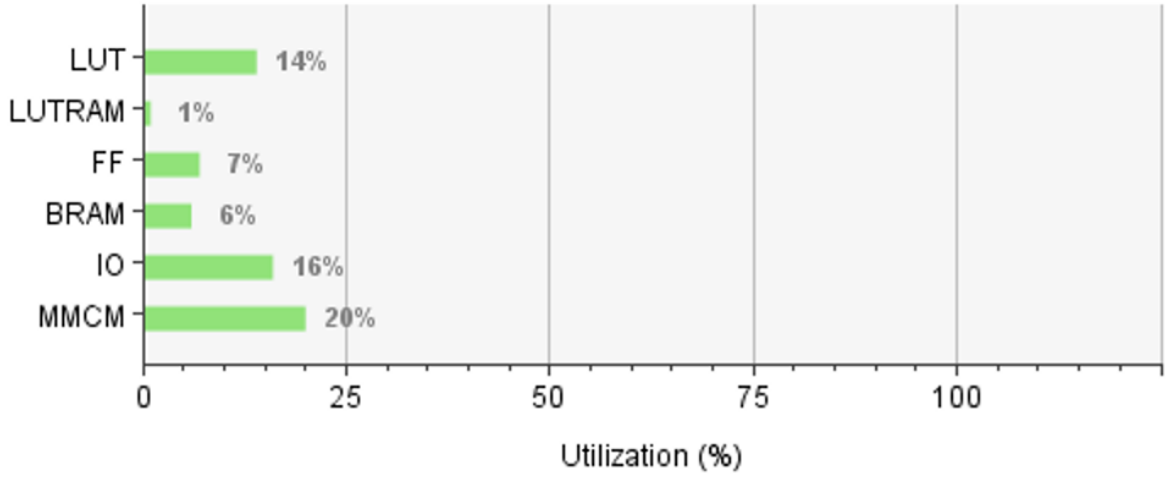
2.2.8: constraints.xdc

```
##============================================================================
## External_Clock_Port
##============================================================================
set_property PACKAGE_PIN W5 [get_ports clk_iport_100MHz]

     set_property IOSTANDARD LVCMOS33 [get_ports clk_iport_100MHz]
     create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5}
[get_ports clk_iport_100MHz]

##============================================================================
## 7 segment display
##============================================================================
set_property PACKAGE_PIN W7 [get_ports {seg_oport[0]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {seg_oport[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg_oport[1]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {seg_oport[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg_oport[2]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {seg_oport[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg_oport[3]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {seg_oport[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg_oport[4]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {seg_oport[4]}]
set_property PACKAGE_PIN V5 [get_ports {seg_oport[5]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {seg_oport[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg_oport[6]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {seg_oport[6]}]
set_property PACKAGE_PIN V7 [get_ports dp_oport]
     set_property IOSTANDARD LVCMOS33 [get_ports dp_oport]
set_property PACKAGE_PIN U2 [get_ports {an_oport[0]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {an_oport[0]}]
set_property PACKAGE_PIN U4 [get_ports {an_oport[1]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {an_oport[1]}]
set_property PACKAGE_PIN V4 [get_ports {an_oport[2]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {an_oport[2]}]
set_property PACKAGE_PIN W4 [get_ports {an_oport[3]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {an_oport[3]}]

##============================================================================
## Pmod Header JA
##============================================================================
```

```
##Sch name = JA1
set_property PACKAGE_PIN J1 [get_ports {spi_cs_oport}]
     set_property IOSTANDARD LVCMOS33 [get_ports {spi_cs_oport}]

##Sch name = JA2
set_property PACKAGE_PIN L2 [get_ports {spi_data_oport}]
     set_property IOSTANDARD LVCMOS33 [get_ports {spi_data_oport}]

##Sch name = JA4
set_property PACKAGE_PIN G2 [get_ports {spi_sclk_oport}]
     set_property IOSTANDARD LVCMOS33 [get_ports {spi_sclk_oport}]

##============================================================================
## Pmod Header JB
##============================================================================
##Sch name = JB1
set_property PACKAGE_PIN A14 [get_ports {midi_iport}]
     set_property IOSTANDARD LVCMOS33 [get_ports {midi_iport}]

##============================================================================
## Implementation Assist
##============================================================================
## These additional constraints are recommended by Digilent, do not
remove!
set_property BITSTREAM.GENERAL.COMPRESS TRUE [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
set_property CONFIG_MODE SPIx4 [current_design]
set_property BITSTREAM.CONFIG.CONFIGRATE 33 [current_design]

set_property CONFIG_VOLTAGE 3.3 [current_design]
set_property CFGBVS VCCO [current_design]
```

2.3: Resource Utilization

Resource	Utilization	Available	Utilization %
LUT	2881	20800	13.85
LUTRAM	2	9600	0.02
FF	2776	41600	6.67
BRAM	3	50	6.00
IO	17	106	16.04
MMCM	1	5	20.00



2.4: Residual Warnings Analysis

No constraints selected for write – common warning that often shows up after synthesis, cleared by Ben as benign.

Port 'reset_iport' is missing in component declaration – we added an additional reset input to tick_generator.vhd. Since not all declarations of the tick generator need the reset input and a default value is provided, this warning is benign.

design project_shell has port dp_oport driven by constant 1 – our seven-segment display implementation for this project didn't require the use of the decimal point.

Found unconnected internal register 'count_reg' and it is trimmed from '16' to '15' bits. – count is declared as a signal in synthesizer.vhd, with an additional bit to account for overflow.

Since the sign bit is always truncated before being passed to the sine LUT, this warning appears and is benign.

design d_to_a has unconnected port dac_data_iport[0] – the sine wave amplitude normalization truncates the first bit of dac_data_iport due to the right bit shift that simulates division by 2. Hence, this warning is expected and benign.

Unused sequential element notes_accumulated_reg was removed – since the notes_accumulated and numnotes variables don't last beyond the lifetime of their process, and are copied to the notes_accumulated_sig and numnotes_sig at the end of that process, Vivado correctly recognizes that an adjacent register has the same value (the variables were declared mainly to simplify arithmetic) and so it doesn't need to maintain a separate register for the variables. Thus, this warning is benign.

The remaining warnings are from the out of context module runs, and come from the Vivado-generated clocking wizard and sine wave LUT.

Appendix 3: Memory Map

Sine Wave LUT

Output width – 12 bits (to match the input of D-to-A converter)

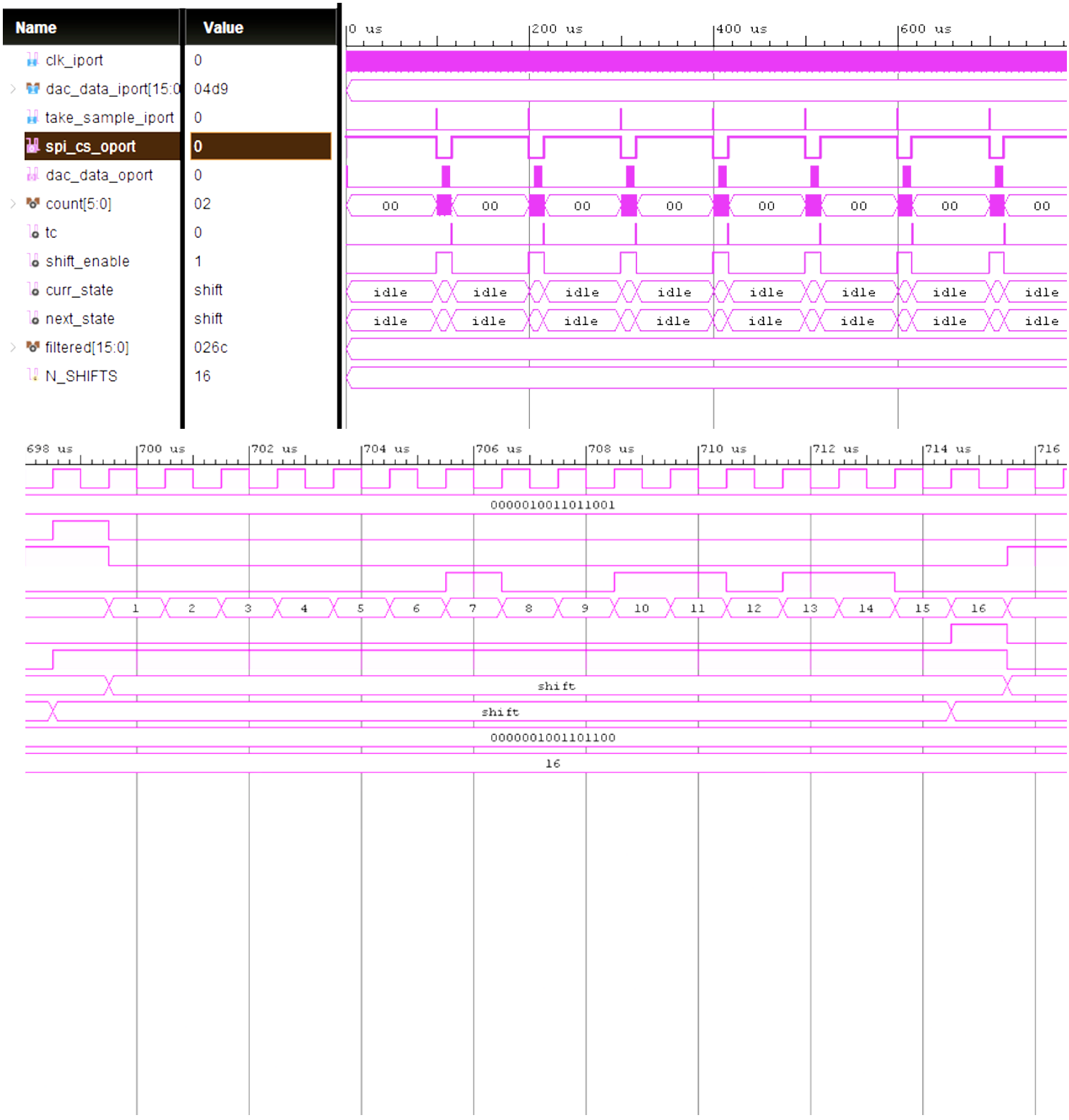
Phase width – 15 bits (48000 Hz / 2^15 = 1.465 Hz base frequency. We needed a relatively small base frequency to accurately distinguish all the notes on the keyboard, particularly for the lower octaves with less separation in frequency between notes.)

The dimensions of this Block ROM are 2^{15} x 12. There are 2^{15} samples stored in the lookup table, each corresponding to the two's complement value of a particular point in a single cycle of a sine wave.

Appendix 4: Waveform Graphs

Simulations

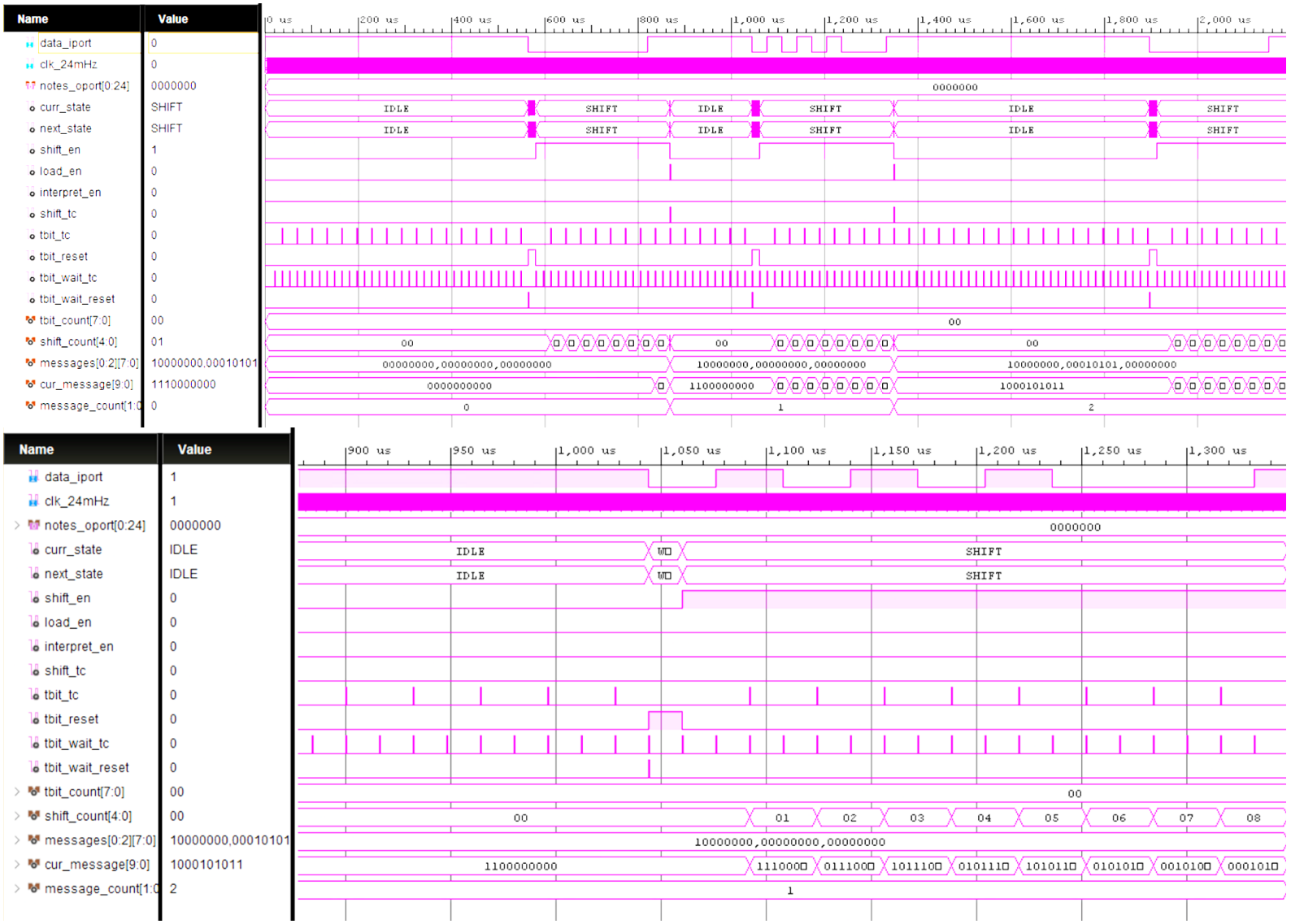
Figure 1: Digital to Analog Controller



The top simulation waveform shows a high-level view of the digital-to-analog controller simulation over roughly 1ms. The bottom waveform provides a detailed view of the state transitions shifting the input into the shift register. The data input remains constant throughout the duration of this testbench.

dac_data_iport – 15-bit parallel data input (in offset binary format, ranging from 0 to 4095).
take_sample_iport – take sample ticks provided by the tick generator. Notice that each state transition occurs after a take_sample tick.
spi_cs_oport – low when sampling. See the Pmod DA2 datasheet for more information.
count – internal signal used for shifting bits into the register. Ranges from 0 to N_SHIFTS (16 in this case).
tc – terminal count, stops the shifting and marks transition to idle state.
filtered – same as the data input, except right-shifted by 1 bit to reduce the total amplitude by half. This is the signal that ultimately becomes the serial dac_data_oport output.

Figure 2: MIDI Interface

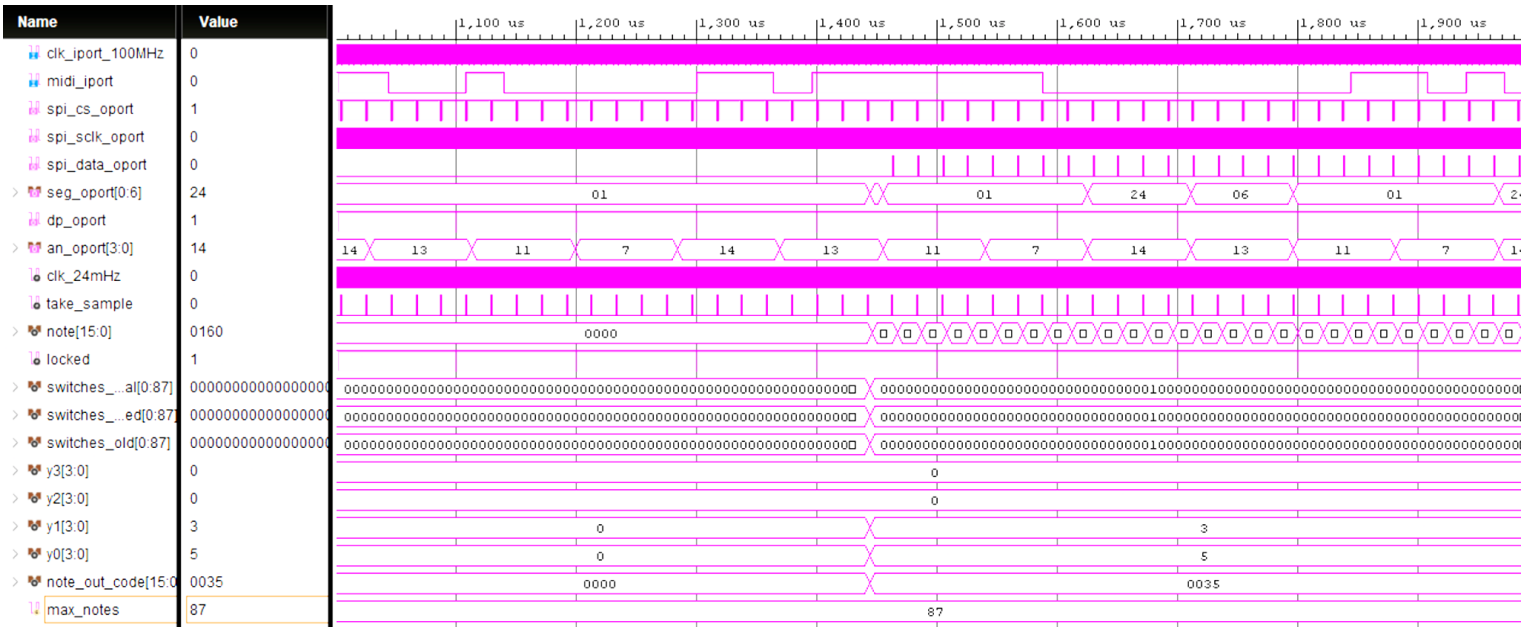


The top diagram is a high level view of the MIDI interface testbench simulation, which simulates MIDI messages for a single note. The bottom diagram provides a more detailed view of the state transitions and the shifting of each bit into the cur_message register.

data_iport – serial data signal provided by the MIDI keyboard.
notes_oport – each bit corresponds to a note. A '1' represents the note at that position being played, whereas a '0' represents a currently inactive note. Note that this only updates after the INTERPRET state concludes.
shift_en, load_en, interpret_en – flags from state machine output, which are high during the SHIFT, LOAD, and INTERPRET states, respectively.
tbit_tc – a tick that goes high every 32 µs (as per MIDI timing protocols).

tbit_wait_tc – a tick that goes high every 16 µs (tbit/2). Used for setting up the beginning of a MIDI message (WAITING state).
tbit_reset, tbit_wait_reset – reset flags for the tbit and tbit/2 tick generators, respectively.
messages – register containing each message that has been read so far. Each MIDI message consists of three bytes, so this register can store up to three bytes before interpreting the message.
cur_message – the current message received from the MIDI keyboard, either being shifted or loaded.

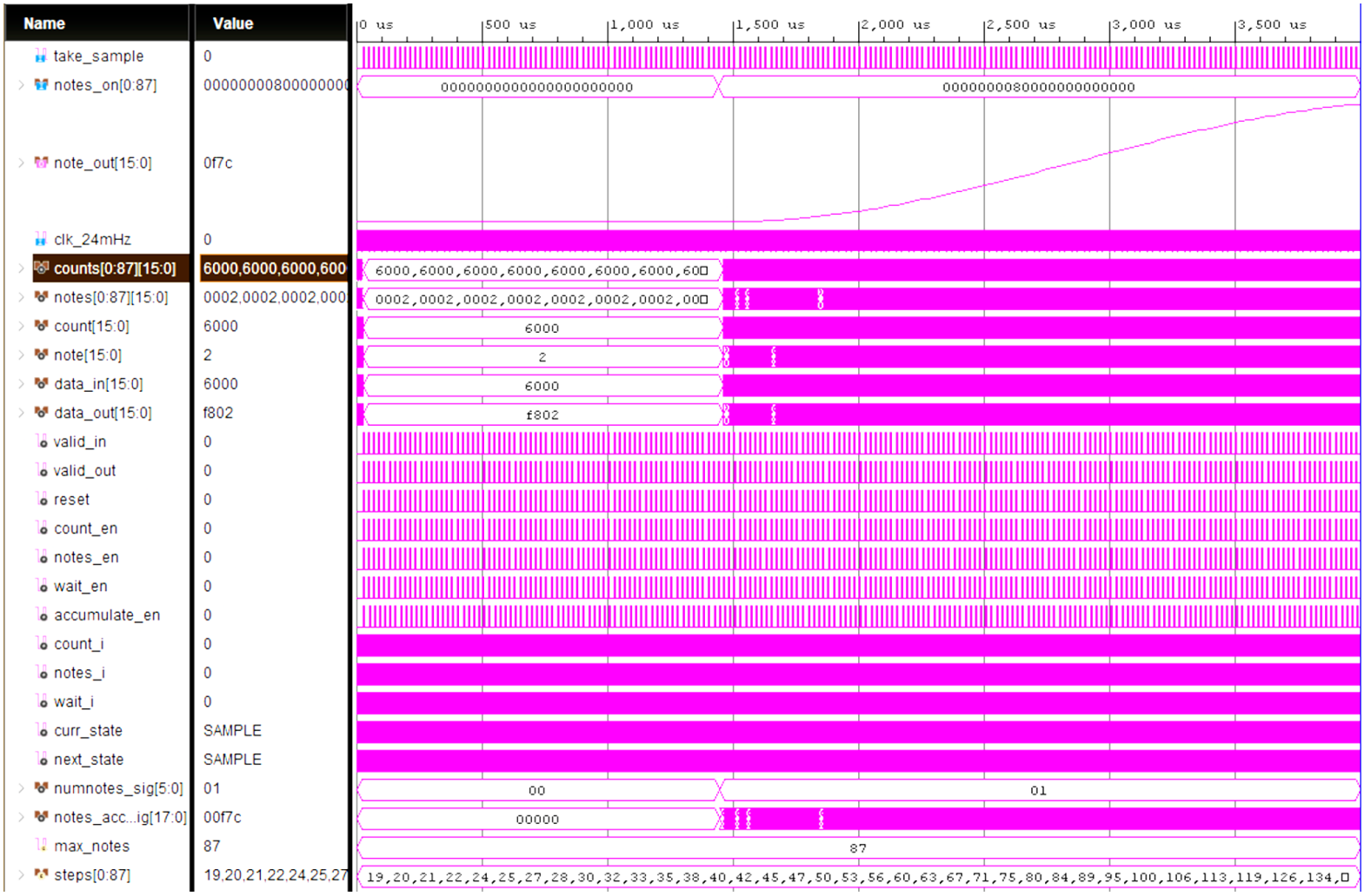
Figure 3: Project Shell



This figure showcases the testbench for the main project shell. It simulates a single note being pressed on the keyboard.

switches_actual, switches_filtered, switches_old – helper signals to avoid a "clicking" noise when depressing a key. There is a process in project_shell.vhd which prevents switches_filtered (corresponding to notes_out) from reflecting the depressed note until the amplitude of the sine wave is near 0, which avoids the abrupt transition associated with stopping the note.
locked – output from system clock generator. Defaults to high once the clock output has been established.
y3, y2, y1, y0 – digits for the 7-segment display. In this diagram, note that **y1** and **y0** match the **note_out_code** of 0035.
note – current value of the sine wave being played.
seg_oport, dp_oport, an_oport – connect to 7-segment display.

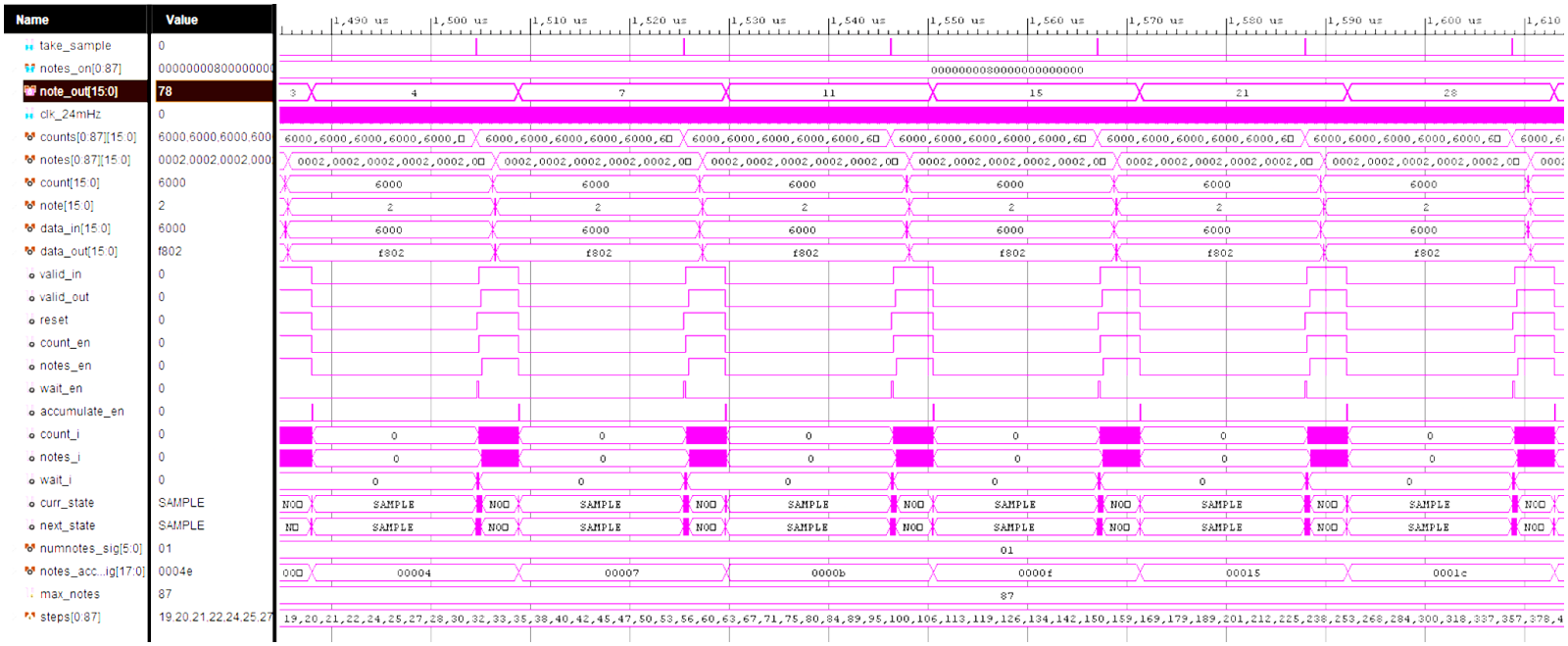
Figure 4: Synthesizer

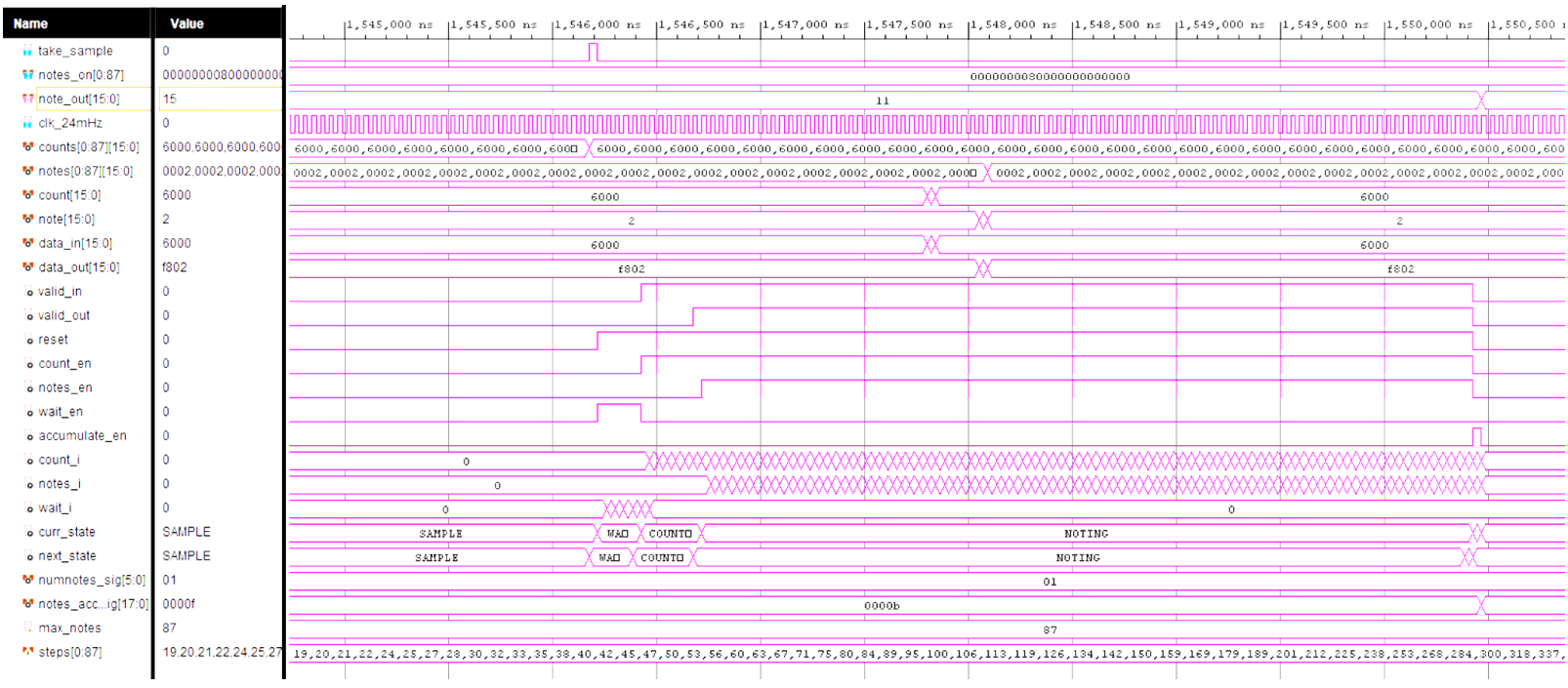


The top diagram displays a high-level view of the synthesizer module replicated with a testbench. The visible waveform is the sine wave eventually converted to a voltage by the digital-to-analog converter. The bottom two diagrams depict increasing levels of detail for each of the state transitions, which are initiated by the take_sample ticks. Once notes_on changes to output a single note, the value of note_out begins increasing, consonant with the pattern of a sine wave with the frequency of the depressed key.

note_out – output note in digital form, to be processed by the D-to-A converter (can include multiple notes synthesized together with polyphony).
counts – ongoing array of counts (the output of each DDS phase accumulator).
notes – ongoing array of notes (the output of the sine wave LUT). Note that both the counts and notes remain at their default values *until* a note is pressed, which avoids the clicking noise associated with the sine wave attack phase.

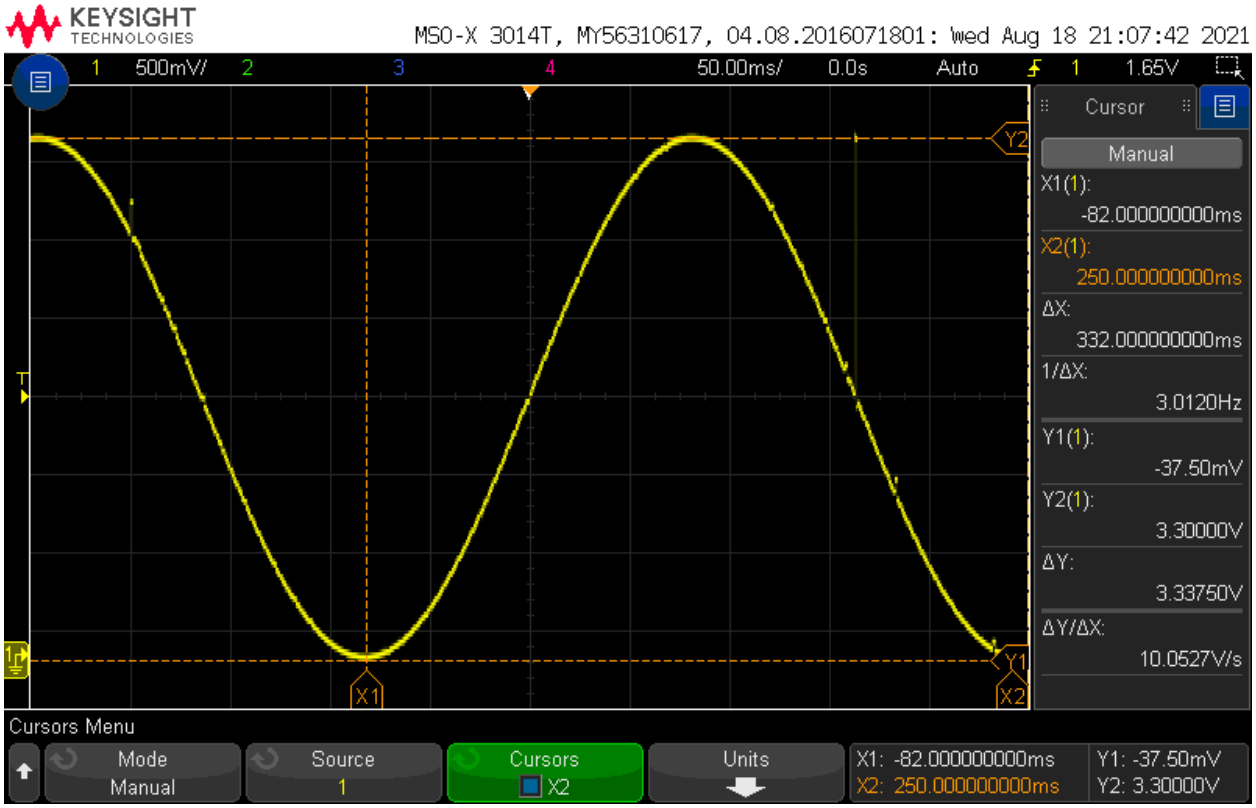
count, note – current count and note, as represented by **count_i** and **note_i** (which iterate from 0 to max_notes each sampling period).
count_en, notes_en, wait_en, accumulate_en – state machine flags.
numnotes_sig, notes_accumulated_sig – represent the number of notes and the total value of each of those notes (not their counts). We divide the sum of the sine waves by the total number of notes pressed to produce polyphony.





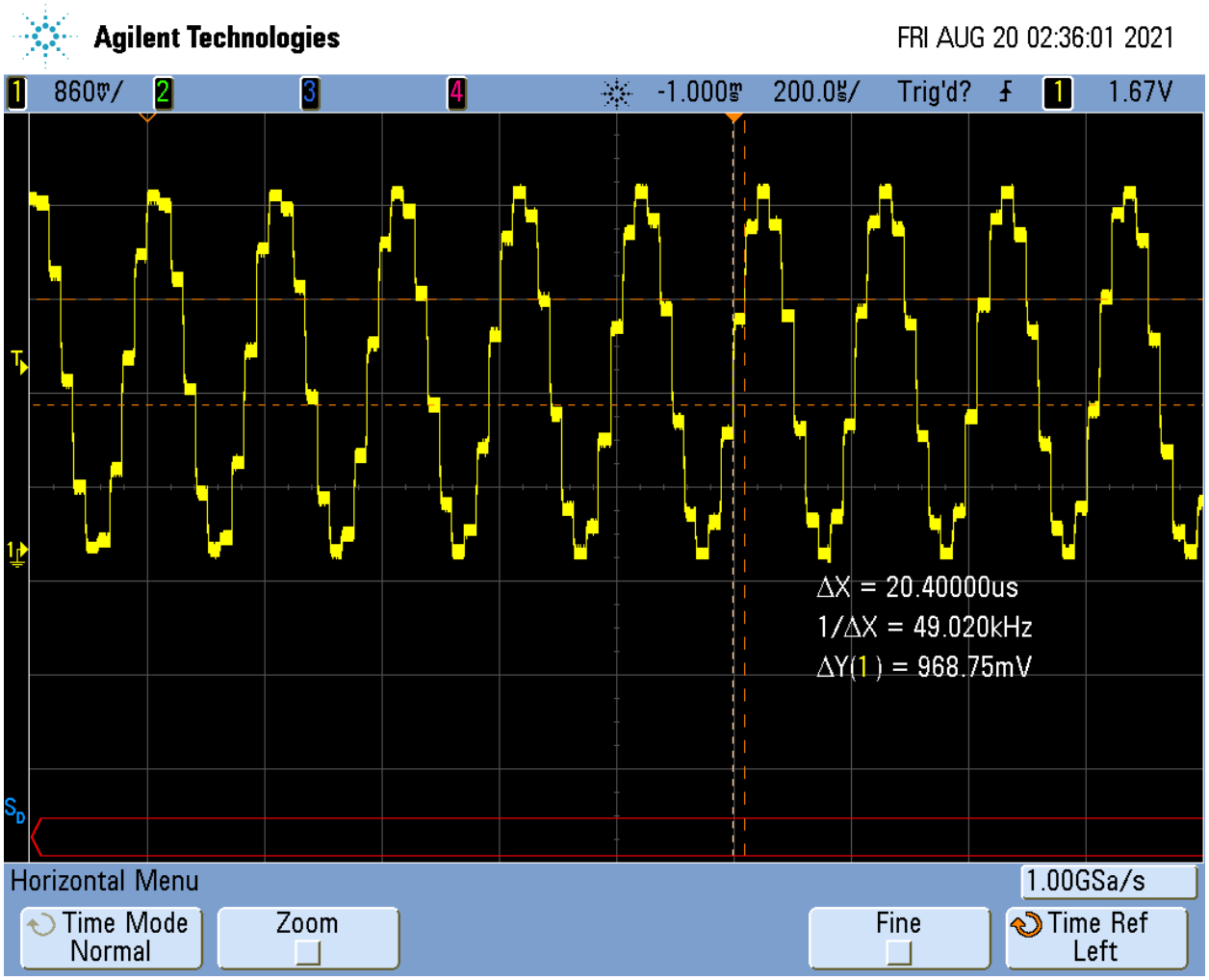
Oscilloscope Output

Figure 1: Digital to Analog Converter Output



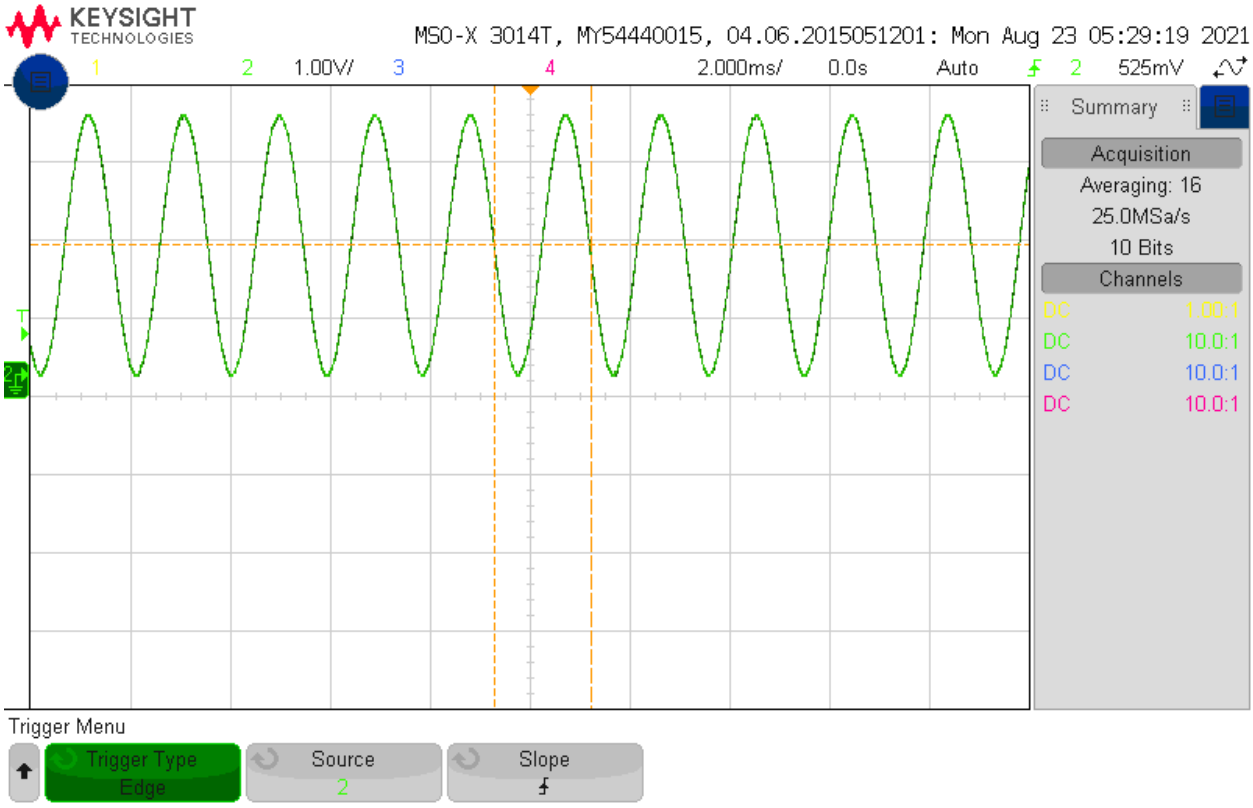
The Oscilloscope screenshot above shows successful voltage generation by the digital to analog converter. This was an important first step for our project's development, as it verified the initial functionality of our digital to analog converter and DDS. It is valuable to note that the voltage represented by the ΔY of the oscilloscope, 3.3 volts, accurately matches the corresponding digital value which we sought to convert.

Figure 2: Erratic Behavior From Shell Testbench during monophonic keyboard testing phase



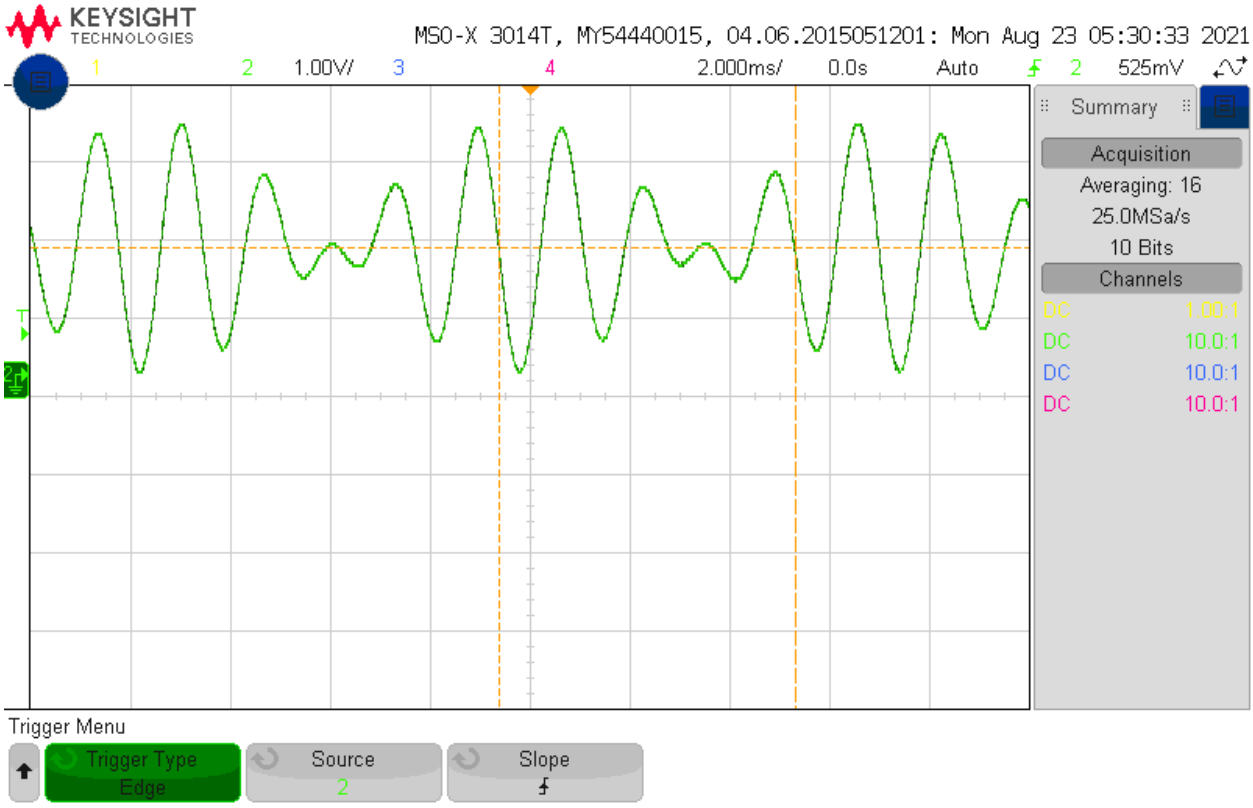
The Oscilloscope screenshot above shows erratic behavior of our shell during the testing phase of our synthesizer block, which contains phase accumulators for each note. The general erratic behavior is what is notable here, which was fixed once we addressed a latch issue in our program.

Figure 3: Successful Monophonic Sine Wave Generation Output



The Oscilloscope screenshot above shows successful sine wave generation by our completed MIDI keyboard system. It is valuable to note that the frequency of the wave matches the note which we pressed, which was 2.16 kHz in this case corresponding to C7.

Figure 4: Successful Polyphonic Sine Wave Generation Output



The Oscilloscope screenshot above shows successful polyphonic wave generation by our MIDI keyboard system. It is valuable to note that after ensuring that the notes start from 0 and end close to 0, the oscilloscope output of multiple keys being pressed at once does not feature any jumps or inconsistencies. Physically, the output of the wave shown above on the oscilloscope is a chord.

Appendix 5: Datasheets

Alesis Q25 MIDI keyboard

ALESIS®

Q25

USB/MIDI KEYBOARD CONTROLLER

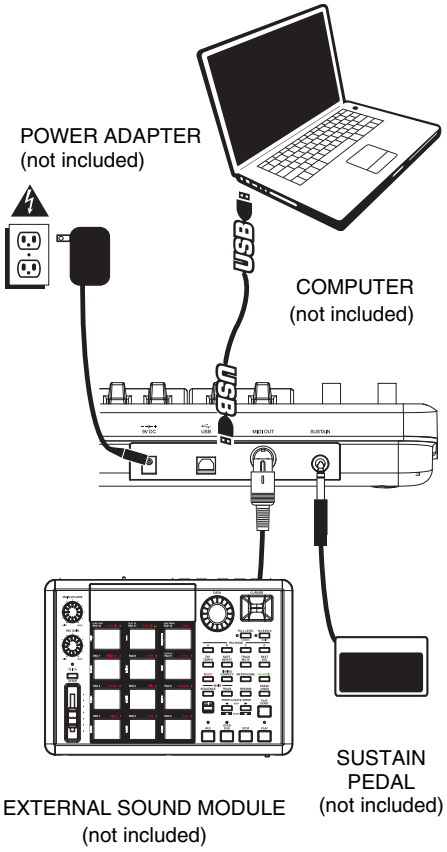
INTRODUCTION

This Quickstart Guide is intended to give you a brief overview of the functionality and features of the Q25. In this manual you will find instructions on how to connect the Q25 and how to use its basic features. Enjoy!

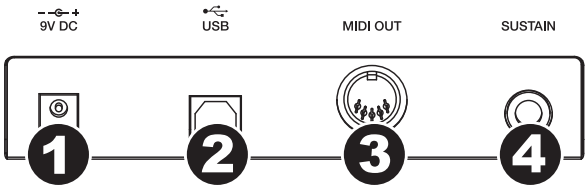
CONNECTION DIAGRAM

Please refer to the following scenario for connecting the Q25.

1. Connect a USB cable from your computer to the Q25. The unit will be powered through the USB connection. Alternatively, if you do not wish to use a computer in your setup or if you wish to power the Q25 externally, please plug in a 9V DC, 500mA power adapter, center-positive, 5.46mm barrel diameter.
2. If you would like to use an external sound module, connect a 5-pin MIDI cable from the KEYBOARD MIDI OUT of the Q25 to the MIDI IN of the external device.

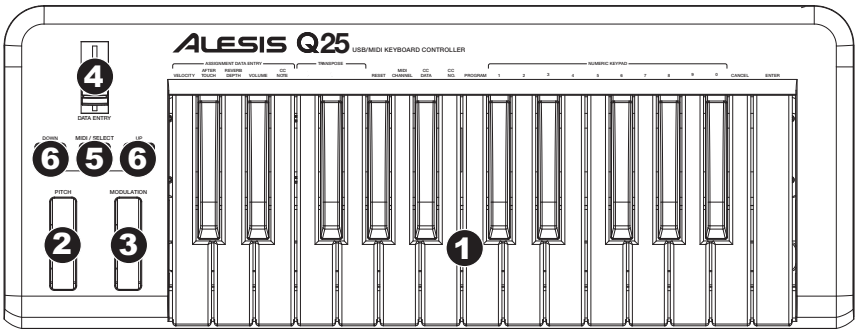


REAR PANEL OVERVIEW



1. **DC POWER ADAPTER INPUT –** Plug in a 9V DC, 500mA power adapter, center-positive, 5.46mm barrel diameter (sold separately) if you do not wish to power the Q25 through the USB connection.
2. **USB CONNECTION –** Plug a standard USB cable into this outlet and into the USB port of your computer. The computer's USB port will provide power to the Q25. This connection is used to send and receive MIDI data to and from your computer and may also be used to send MIDI data from your computer to a device attached to the MIDI OUT port of the Q25.
3. **MIDI OUT –** Use a five-pin MIDI cable (sold separately) to connect this jack to the MIDI IN of an external device.
4. **SUSTAIN PEDAL INPUT –** Connect a 1/4" TS sustain pedal (sold separately) to this input.

TOP PANEL OVERVIEW



1. **KEYBOARD –** The KEYBOARD functions as a normal electronic piano keyboard during performance but can also be used to adjust MIDI settings and send MIDI messages. The labels above the keys indicate their functions. The numbered keys allow you to enter values for settings. Press the Cancel or Enter key to cancel or confirm your selection, respectively. See SELECTING & EDITING MIDI COMMANDS for more information.
2. **PITCH BEND WHEEL –** Transmits MIDI Pitch Bend information to raise or lower the pitch of a note temporarily.
3. **MODULATION WHEEL –** This wheel can be used to transmit continuous controller data (CC #1 or Modulation Depth).
4. **DATA ENTRY SLIDER** – This slider lets you send MIDI messages for the currently selected parameter from the KEYBOARD (e.g., REVERB DEPTH, VOLUME, etc.). You can select a parameter by pressing MIDI / SELECT then the corresponding key on the KEYBOARD.
5. **MIDI / SELECT –** Pressing this button allows you to adjust MIDI settings and send MIDI messages by pressing labeled keys on the KEYBOARD. See SELECTING & EDITING MIDI COMMANDS for more information.
6. **OCTAVE UP / DOWN –** These buttons can be used to shift the keyboard's range up and down.

SELECTING & EDITING MIDI COMMANDS

The MIDI / SELECT button and the keys on the KEYBOARD allow you to adjust MIDI parameters and send precise MIDI messages and information quickly and easily:

1. **Press MIDI / SELECT.**
2. **Press the key on the KEYBOARD whose MIDI setting (printed above the key) you want to adjust or send.**
3. **Enter a value with the numbered keys on the KEYBOARD.**
4. **Press the Enter or Cancel key on the KEYBOARD to confirm or cancel your choice, respectively.**
5. **Press MIDI / SELECT if the button is still lit.**

VELOCITY – Press this key to set the DATA ENTRY SLIDER to adjust the note velocity. Increasing this value increases the MIDI velocity value generated by a light key press.

AFTERTOUCH – Press this key to set the DATA ENTRY SLIDER to send Aftertouch information (also sometimes referred to as Channel Pressure).

REVERB DEPTH – Press this key to set the DATA ENTRY SLIDER to CC #91 (Reverb Send Level). The default value for this setting is 64.

VOLUME – Press this key to set the DATA ENTRY SLIDER to CC #7 (Channel Volume).

CC DATA (Control Change Data) **–** Sets the value to be sent. Pressing the Enter key will send a MIDI message.

OCTAVE – Lowers or raises the octave range of the KEYBOARD with the "OCTAVE –" and "OCTAVE +" buttons, respectively. You can transpose the keyboard up to two octaves in either direction. Press STANDARD to return the KEYBOARD to its original octave range

b / # – Lowers or raises (respectively) the pitch of the entire KEYBOARD a semitone.

RESET – Press this key followed by the Enter key to send a reset message to all controllers and return the Q25 to its original settings.

Note: This button does not reset the current octave range or transposition of the KEYBOARD.

CC NO. (Control Change Number) **–** Sets the MIDI CC# to be sent. (A message will not be sent until the Enter key is pressed when selecting CC Data.)

PROGRAM – Sets the MIDI Program Change number.

Note: *After you enter a CC#, the number will be remembered by the Q25. However, if you want to send a CC or Program Change message, you must enter the value using the numbered keys on the KEYBOARD each time (even if the desired value is shown in the LCD) before pressing Enter.*

TROUBLESHOOTING

PROBLEM	CAUSE	SOLUTION
The display does not light up.	No power.	Please make sure that the Q25 is connected to your computer and that the computer is powered on.
		If using a power adapter, please make sure that the adapter is plugged into a live power outlet.
No sound from target device.	Q25 not properly connected.	Check your computer's USB connection to confirm that the Q25 is recognized. If necessary, replug the connection and restart your computer.
		If controlling an external hardware module, make sure that the MIDI cable is connected from the Q25 to the device's MIDI IN port.
	Q25 connected after software application has started.	Restart the software application with the controller plugged in.
	Problem is caused by use of a USB hub.	Unplug the Q25 from the USB hub and connect directly to the computer.
	Software application not set to receive MIDI data from the Q25.	Ensure that the Q25 or USB MIDI device is listed as an active MIDI source in your application. Usually, the MIDI settings can be accessed through the application's Preferences menu.
	Q25's MIDI channel not the same as application's incoming MIDI chanel.	Make sure that the Q25 is sending MIDI information on the channel that the target device expects.
Notes sustain continuously.	Sustain pedal was plugged in after the Q25 was powered on.	Turn the unit's power off, wait a moment and then turn it on again.
	Stuck notes due to incomplete MIDI data.	Turn the unit's power off, wait a moment and then turn it on again.
Sustain pedal works in reverse.	Sustain pedal was plugged in after power was turned on.	With the pedal plugged in, turn the unit's power off, wait a moment, and turn it on again.

SPECIFICATIONS

POWER:	USB, 9V DC, 500mA, center-positive, 5.46mm barrel diameter (sold separately)
KEYBOARD:	25 keys
ACCESSORIES:	Quickstart Guide, USB cable
MIDI OUTPUTS:	1 5-pin jack
USB:	1 slave connector (MIDI over USB)

http://www.alesis.com/q25

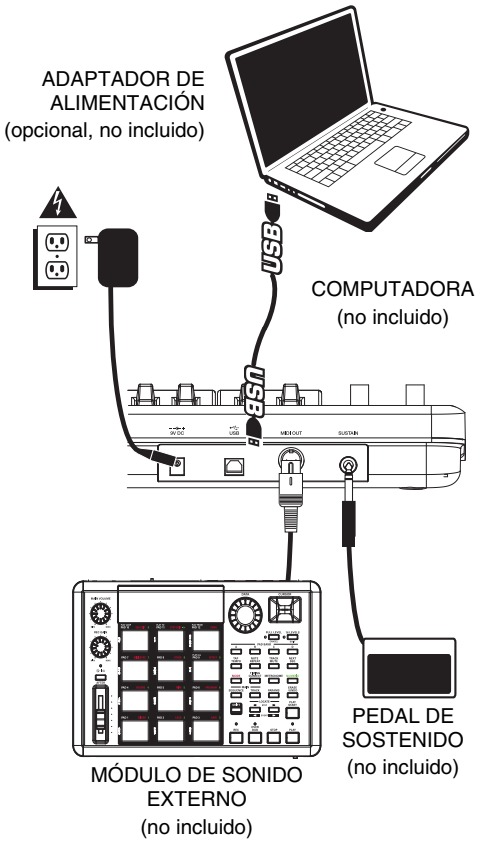
INTRODUCCIÓN

Este Manual de inicio rápido tiene la finalidad de brindarle una breve descripción general de la funcionalidad y las características del Q25. Encontrará en el mismo instrucciones sobre cómo conectar el Q25 y cómo usar sus características básicas. ¡Que lo disfrute!

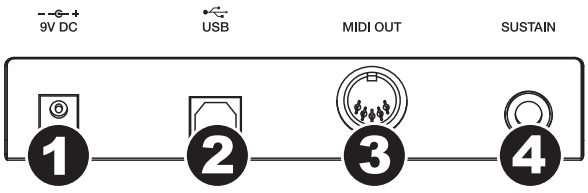
DIAGRAMA DE CONEXIÓN

Consulte el siguiente escenario para conectar el Q25.

1. Conecte un cable USB de su computadora al Q25. La unidad se alimenta por la conexión USB. Como alternativa, si no desea usar una computadora en su configuración o desea alimentar el Q25 externamente, enchufe un adaptador de alimentación de 9 V CC, 500 mA, centro postigo, diámetro 5.46mm (no incluido).
2. Si desea usar un módulo de sonido externo, conecte un cable MIDI de 5 pines desde MIDI OUT (Salida MIDI) del Q25 a la ENTRADA MIDI del dispositivo externo.

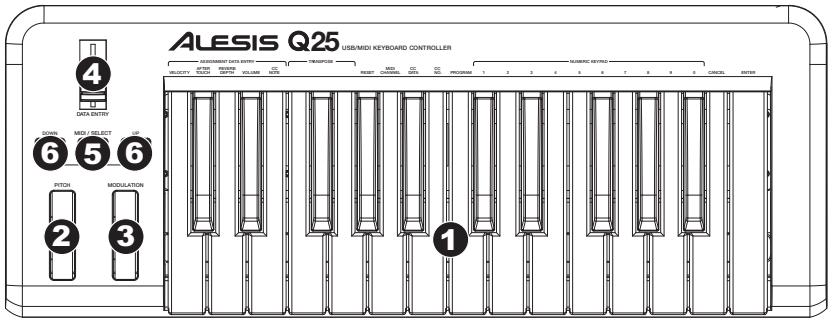


VISTA DEL PANEL TRASERO



1. **ENTRADA DEL ADAPTADOR DE ALIMENTACIÓN DE CC –** Para enchufar un adaptador de alimentación de 9 V CC, 500 mA, centro postigo, diámetro 5.46mm (vendido por separado) si no desea alimentar el Q25 a través de la conexión USB.
2. **CONEXIÓN USB -** Enchufe un cable USB estándar en este conector y en el puerto USB de su computadora. El puerto USB de la computadora proporciona alimentación eléctrica al Q25. Esta conexión se usa para enviar y recibir datos MIDI hacia y desde la computadora y se puede usar también para enviar datos MIDI desde la computadora a un dispositivo conectado al puerto MIDI OUT (Salida MIDI) del Q25.
3. **SALIDA MIDI –** Use un cable MIDI estándar de cinco pines (vendido por separado) para conectar este jack a la ENTRADA MIDI de un dispositivo externo.
4. **ENTRADA DE PEDAL DE SOSTENIDO –** Conecte un pedal de sostenido TS de 1/4" (vendido por separado) a esta entrada.

VISTA DEL PANEL SUPERIOR



1. **TECLADO –** El TECLADO funciona como un teclado de piano electrónico normal durante la interpretación pero se puede usar también para ajustar parámetros MIDI y mandar mensajes MIDI. Los rótulos que están arriba de las teclas indican sus funciones. Las teclas numeradas permiten introducir valores para los parámetros. Pulse las teclas Cancel o Enter para cancelar o confirmar su selección, respectivamente. Para más información, consulte CÓMO SELECCIONAR Y EDITAR COMANDOS MIDI.
2. **RUEDA DE INFLEXIÓN DE PITCH –** Transmite información de inflexión de pitch MIDI para elevar o bajar el pitch de una nota temporalmente.
3. **RUEDA DE MODULACIÓN –** Esta rueda se puede usar para transmitir datos continuos del controlador (CC #1 o profundidad de modulación).
4. **CURSOR DE ENTRADA DE DATOS** – Este cursor permite enviar mensajes MIDI correspondientes al parámetro seleccionado en ese momento desde el TECLADO (por ej., REVERB DEPTH (Profundidad de reverberación), VOLUME (Volumen), etc.). Es posible seleccionar un parámetro pulsando MIDI / SELECT (MIDI / Selección) y luego la tecla correspondiente del TECLADO.
5. **MIDI / SELECCIÓN –** Pulsando este botón, es posible ajustar parámetros MIDI y enviar mensajes MIDI pulsando las teclas rotuladas del TECLADO. Para más información, consulte CÓMO SELECCIONAR Y EDITAR COMANDOS MIDI.
6. **OCTAVA ARRIBA / ABAJO –** Estos botones se pueden usar para desplazar la gama del teclado hacia arriba y abajo.

COMO SELECCIONAR Y EDITAR COMANDOS MIDI

El botón MIDI / SELECT y las teclas del TECLADO permiten ajustar parámetros MIDI y enviar mensajes e información MIDI precisos rápida y fácilmente:

1. **Pulse MIDI / SELECT.**
2. **Pulse la tecla del TECLADO cuyo parámetro MIDI (impreso arriba de la tecla) desea ajustar o enviar.**
3. **Introduzca un valor con las teclas numeradas del TECLADO.**
4. **Pulse la tecla Enter o Cancel del TECLADO para confirmar o cancelar su elección, respectivamente.**
5. **Pulse MIDI / SELECT si el botón está iluminado.**

VELOCIDAD – Pulse esta tecla para configurar el CURSOR DATA ENTRY para ajustar la velocidad de nota. Al aumentar este valor se incrementa el valor de velocidad MIDI generado por una pulsación ligera de la tecla.

POST-PULSACIÓN – Pulse esta tecla para configurar el CURSOR DATA ENTRY (Entrada de datos) para enviar información de post-pulsación (también llamada a veces "presión del canal").

PROFUNDIDAD DE REVERBERACIÓN – Pulse esta tecla para configurar el CURSOR DATA ENTRY a CC #91 (Nivel de envío de reverberación). El valor predeterminado de este parámetro es 64.

VOLUMEN – Pulse esta tecla para configurar el CURSOR DATA ENTRY a CC #7 (Volumen de canal).

DATOS DE CC (Datos de cambio de control) **–** Establece el valor a enviar. Al pulsar la tecla Enter, se envía el mensaje MIDI.

b / # – Baja o eleva (respectivamente) el pitch del TECLADO completo un semitono.

REINICIACIÓN – Pulse esta tecla y a continuación la tecla Enter para enviar un mensaje de reiniciación a todos los controladores y regresar el Q25 a sus valores de parámetros originales.

Nota: Este botón no reinicia la gama de octavas actual o la transposición del TECLADO.

Nº DE CC (Número de cambio de control) **–** Establece el CC# MIDI a enviar. (No se envía el mensaje hasta que se pulse la tecla Enter cuando se seleccionan datos de CC.)

PROGRAMA – Establece el número de cambio de programa MIDI.

Nota: *Después de introducir un CC#, el Q25 recuerda el número. En cambio, si desea enviar un mensaje de CC o cambio de programa, debe introducir el valor con las teclas numeradas del TECLADO cada vez (aunque el valor deseado aparezca en la LCD) antes de pulsar Enter.*

SOLUCIÓN DE PROBLEMAS

PROBLEMA	CAUSA	SOLUCIÓN
La pantalla no se ilumina.	No hay alimentación.	Asegúrese de que el Q25 esté conectado a la computadora y que ésta esté encendida.
		Si usa un adaptador de alimentación, asegúrese de que el mismo esté enchufado a un tomacorriente alimentado.
No hay sonido del dispositivo destinatario.	Q25 conectado incorrectamente.	Verifique las conexiones USB de su computadora para confirmar que el Q25 sea reconocido. Si fuera necesario, enchufe nuevamente la conexión y reinicie la computadora.
		Si está controlando un módulo de hardware externo, asegúrese de que el cable MIDI esté conectado del Q25 al puerto MIDI IN del dispositivo.
	Q25 conectado después de iniciar la aplicación de software.	Reinicie la aplicación de software con el controlador enchufado.
	Problema causado por usar un concentrador (hub) USB.	Desenchufe el Q25 del concentrador USB y conéctelo directamente a la computadora.
	La aplicación de software no está configurada para recibir datos MIDI desde el Q25.	Asegúrese de que el Q25 o el dispositivo MIDI "USB" esté clasificado como fuente de MIDI activa en su aplicación. Normalmente, se puede acceder a los parámetros MIDI a través del menú Preferentes (Preferencias) de la aplicación.
	El canal MIDI del Q25 no es igual al canal MIDI de entrada de la aplicación.	Asegúrese de que el Q25 esté enviando datos MIDI en el canal esperado por el dispositivo destinatario.
Las notas se sostienen de manera constante.	El pedal de sostenido fue enchufado después de encender la unidad.	Apague la unidad, espere un momento y enciéndala otra vez.
	Notas pegadas debido a datos MIDI incompletos.	Apague la unidad, espere un momento y enciéndala otra vez.
El pedal de sostenido funciona de manera inversa.	El pedal de sostenido fue enchufado después de encender la unidad.	Con el pedal enchufado, apague la unidad, espere un momento y enciéndala otra vez.

ESPECIFICACIONES TÉCNICAS

ALIMENTACIÓN: USB, 9 V CC, 500 mA, centro positive, diámetro 5.46mm (vendido por separado)
TECLADO: 25 teclas
ACCESORIOS: Guía de inicio rápido, cable USB
SALIDAS MIDI: 1 jack a 5 pines
USB: 1 conector esclavo (MIDI por USB)

http://www.alesis.com/q25

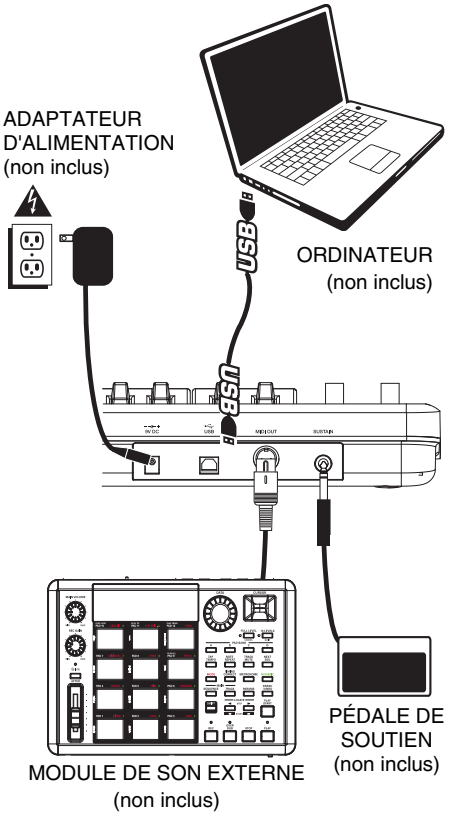
INTRODUCTION

Ce guide d'utilisation simplifié vous propose une vue d'ensemble des fonctions et des caractéristiques du Q25. Vous trouverez dans ce guide les instructions de raccordement et d'utilisation du Q25. Amusez-vous bien!

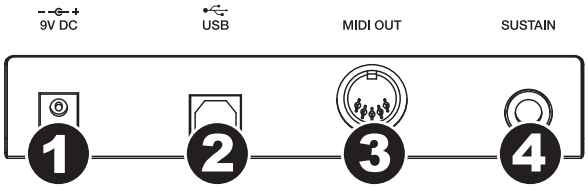
SCHÉMA DE CONNEXION

Veuillez vous reporter aux consignes suivantes pour le raccordement du Q25.

1. Branchez un câble USB de l'ordinateur au Q25. L'appareil sera alimenté par la connexion USB. Cependant, si vous ne désirez pas utiliser un ordinateur ou si vous préférez que le Q25 soit alimenté directement, vous pouvez utiliser un adaptateur d'alimentation 9 V c.c 500 mA à centre positif, diamètre 5,46mm (vendu séparément).
2. Si vous désirez utiliser un module de son externe, branchez un câble MIDI à 5 broches de la sortie MIDI (MIDI OUT) du clavier à l'entrée MIDI de l'appareil externe.

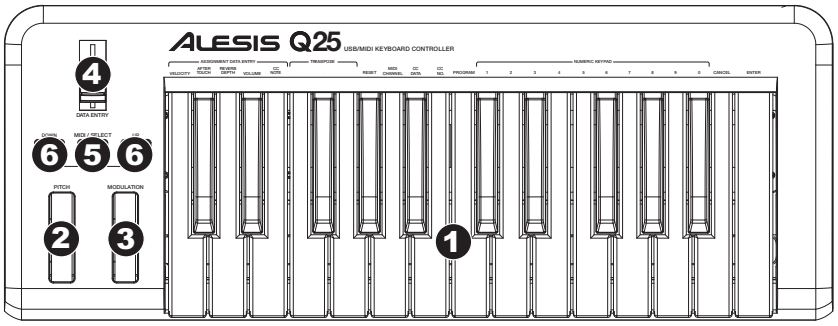


CARACTÉRISTIQUES DU PANNEAU ARRIÈRE



1. **ENTRÉE D'ALIMENTATION –** Branchez un adaptateur 9 V c.c. 500 mA à centre positif, diamètre 5,46mm (vendu séparément) si vous ne désirez pas alimenter le Q25 via la connexion USB.
2. **CONNEXION USB –** Branchez un câble USB standard dans cette sortie et l'autre extrémité dans le port USB d'un ordinateur. Le port USB de l'ordinateur permet d'alimenter le Q25. Cette connexion sert à envoyer et recevoir des données MIDI de votre ordinateur et peut être utilisée pour envoyer des données MIDI de votre ordinateur à un appareil externe branché à la sortie (MIDI OUT) du Q25.
3. **SORTIE MIDI -** Vous pouvez brancher un câble MIDI à cinq broches (vendu séparément) à cette sortie et à l'entrée MIDI IN d'un appareil externe. .
4. **ENTRÉE DE PÉDALE DE SOUTIEN (SUSTAIN) –** Cette entrée permet de brancher une pédale TS de 1/4 po (vendue séparément).

CARACTÉRISTIQUES DU PANNEAU SUPÉRIEUR



1. **CLAVIER –** Le clavier fonctionne comme un clavier électronique ordinaire durant les prestations, mais peut également être utilisé pour ajuster les paramètres MIDI et envoyer des messages MIDI. Les étiquettes au dessus des touches indiquent leurs fonctions. Les touches numérotées vous permettent d'entrer la valeur des paramètres. Appuyez sur la touche Cancel ou Enter pour annuler ou confirmer votre sélection. Reportez-vous à la section SÉLECTION ET MODIFICATIONS DES COMMANDES MIDI pour plus d'informations.
2. **MOLETTE DE MODULATION TEMPORAIRE DE LA VITESSE DE LECTURE –** Cette molette permet d'envoyer des données MIDI concernant l'augmentation ou la réduction temporaire de la vitesse de lecture.
3. **MOLETTE DE MODULATION –** Cette molette peut être utilisée pour transmettre des données de contrôleur en continu ((CC no. 1 ou Modulation Depth).
4. **POTENTIOMÈTRE D'ENTRÉE DES DONNÉES –** Ce potentiomètre permet d'envoyer des messages MIDI pour le paramètre sélectionné à partir du clavier (p. ex., REVERB DEPTH, VOLUME, ETC.).Vous pouvez sélectionné le paramètre en appuyant sur MIDI / SELECT et en appuyant sur la touche correspondante du clavier.
5. **TOUCHE MIDI / SELECT –** Cette touche permet de régler les paramètres MIDI et d'envoyer des messages MIDI en appuyant sur les touches identifiées sur le clavier. Reportez-vous à la section SÉLECTION ET MODIFICATIONS DES COMMANDES MIDI pour plus d'informations.
6. **TOUCHES OCTAVE UP / DOWN –** Ces touches permettent d'augmenter ou de diminuer la plage d'octaves du clavier.

SÉLECTION ET MODIFICATION DES COMMANDES MIDI

La touche MIDI / SELECT et les touches du clavier permettent de régler les paramètres MIDI et d'envoyer des messages et des données MIDI précis rapidement et facilement.

1. **Appuyez sur la touche MIDI / SELECT.**
2. **Appuyez sur la touche du clavier correspondant au paramètre MIDI (identifiée au dessus de la touche) que vous désirez modifier ou envoyer.**
3. **Entrez la valeur à l'aide des touches numériques du clavier.**
4. **Appuyez sur la touche Enter ou Cancel du clavier afin de confirmer ou d'annuler votre sélection.**
5. **Appuyez sur la touche MIDI / SELECT si la touche est toujours allumée.**

VELOCITY – Appuyez sur cette touche pour programmer le potentiomètre d'entrée de données (DATA ENTRY) afin de régler la dynamique de la note. Augmenter cette valeur permet d'augmenter la valeur dynamique MIDI produite avec un touché léger.

AFTERTOUCH – Appuyez sur cette touche pour programmer le potentiomètre d'entrée de données (DATA ENTRY) afin d'envoyer de l'information concernant l'Aftertouch (ou la pression de canal).

REVERB DEPTH – Appuyez sur cette touche pour programmer le potentiomètre d'entrée de données (DATA ENTRY) à CC no. 91 (niveau d'émission de réverbération). La valeur par défaut est 64.

VOLUME – Appuyez sur cette touche pour programmer le potentiomètre d'entrée de données (DATA ENTRY) à CC no. 7 (volume du canal).

CC DATA (Control Change Data) **–** Cette touche permet de régler la valeur des données de changement de commande à envoyer. Appuyer sur la touche Enter permet d'envoyer un message MIDI.

b / # – Cette touche permet d'augmenter ou de diminuer la tonalité du clavier d'un demi-ton.

RESET – Appuyez sur cette touche puis sur la touche Enter permet d'envoyer un message à tous les contrôleurs et de réinitialiser les paramètres par défaut du Q25.
Remarque : Cette touche ne réinitialise pas la plage d'octaves ou la transposition du clavier.

CC NO. (Control Change Number) **–** Cette touche permet de programmer le numéro CC MIDI à envoyer (numéro de changement de commande). (Le message n'est envoyé que si la touche Enter est enfoncée lorsque vous sélectionnez les données CC.)

PROGRAM – Cette touche permet de régler le numéro de changement de programme.

Remarque : *Lorsque vous entrez un numéro CC, le Q25 le sauvegarde. Cependant, si vous désirez envoyer un message CC ou de changement de programme, vous devez entrer la valeur à l'aide des touches numérotées sur le clavier chaque fois (même si la valeur est affichée à l'écran) avant d'appuyer sur la touche Enter.*

GUIDE DE DÉPANNAGE

PROBLÈME	CAUSE	SOLUTION
L'écran d'affichage ne s'allume pas.	Aucune alimentation.	Assurez-vous que le Q25 est correctement branché au port USB de votre ordinateur et que l'ordinateur est sous tension.
		Si vous utilisez un adaptateur, veuillez vous assurer qu'il est branché dans une prise de courant sous tension.
Aucun son provenant de l'appareil cible.	Le Q25 est mal branché.	Vérifiez les connexions USB de votre ordinateur pour vous assurer que le Q25 est reconnu par votre ordinateur. Il peut s'avérer nécessaire de débrancher puis rebrancher les appareils et de relancer votre ordinateur.
		Si vous commandez un module externe matériel, assurez-vous que le câble MIDI est branché du Q25 à l'entrée MIDI de l'appareil.
	Le Q25fut branché après que le logiciel soit lancé.	Relancez le logiciel seulement après avoir branché le Q25.
	Problèmes causés par l'utilisation d'un répéteur USB.	Essayez de débrancher le Q25 du répéteur USB et de le brancher directement à l'ordinateur.
	L'application logicielle n'est pas configurée pour recevoir des données MIDI provenant du Q25.	Assurez-vous que le Q25, ou le dispositif USB MIDI, est inscrit comme source active dans votre application. Habituellement, les paramètres MIDI sont accessibles via le menu Preferences de l'application.
	Le canal MIDI du Q25 n'est pas le même que celui d'entrée MIDI de l'application.	Assurez-vous que le Q25 achemine les données MIDI par le canal d'arrivée de l'appareil.
Les notes sont maintenues de façon continue.	La pédale de soutien (Sustain) a été branchée après que l'appareil a été mis sous tension.	Mettez l'appareil hors tension, attendez quelques secondes et remettez-le sous tension.
	Certaines notes sont bloquées parce que les données MIDI sont incomplètes.	Mettez l'appareil hors tension, attendez quelques secondes et remettez-le sous tension.
La pédale de soutien (sustain) fonctionne à l'envers.	La pédale de soutien (Sustain) a été branchée après que l'appareil a été mis sous tension.	Branchez la pédale, puis mettez l'appareil hors tension, attendez quelques secondes et remettez-le sous tension.

SPÉCIFICATIONS

ALIMENTATION ÉLECTRIQUE : USB, 9 V c.c., 500 mA, à centre positif, diamètre 5,46mm (vendu séparément)
CLAVIER : 25 touches
ACCESSOIRES : Guide d'utilisation simplifié, câble USB
SORTIES MIDI : 1 entrée à cinq broches
USB : 1 connecteur esclave (MIDI sur USB)

http://www.alesis.com/q25

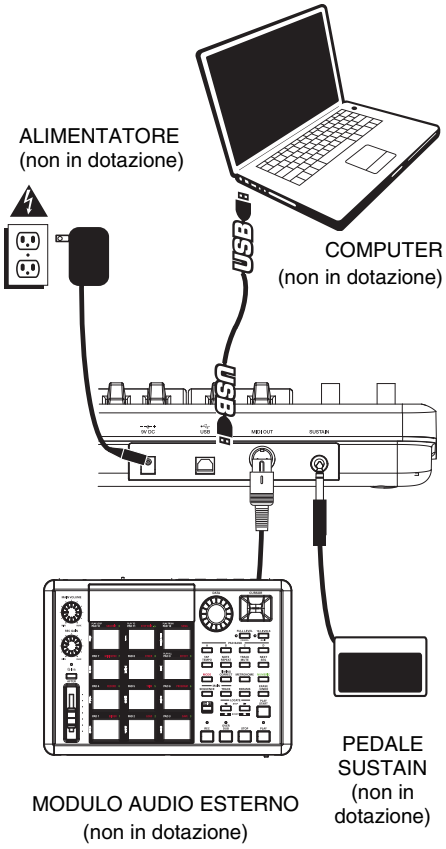
INTRODUZIONE

Questa guida rapida di utilizzo è intesa a fornire una breve panoramica sulle funzioni e caratteristiche del Q25. In questo manuale troverete le istruzioni su come collegare il Q25 e su come servirsi delle sue funzioni base. Buon divertimento!

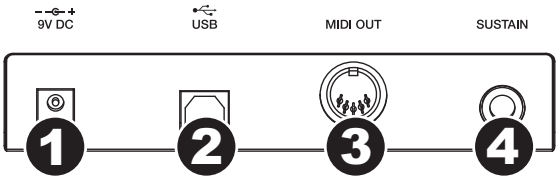
SCHEMA DEI COLLEGAMENTI

Per collegare il Q25, fare riferimento al seguente caso:

1. Collegare un cavo USB dal computer al Q25. L'apparecchio verrà alimentato tramite il collegamento USB. Alternativamente, se non si desidera utilizzare un computer nell'impianto o se si desidera alimentare il Q25 esternamente, inserire un adattatore CC di alimentazione da 9V, 500mA, positivo centrale, diametro 5,46mm (venduto separatamente).
2. Se si desidera utilizzare un modulo audio esterno, collegare un cavo MIDI a 5 poli dall'USCITA TASTIERA MIDI del Q25 all'ingresso MIDI IN del dispositivo esterno.

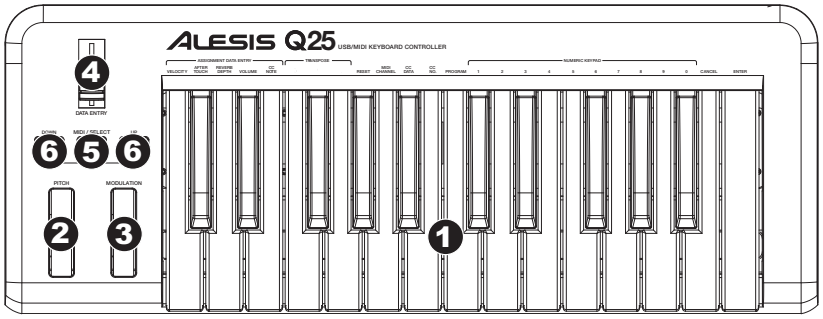


PANORAMICA PANNELLO POSTERIORE



1. **INGRESSO ADATTATORE DI ALIMENTAZIONE CC –** Inserire un adattatore di alimentazione CC da 9V, 500mA, positivo centrale, diametro 5,46mm (venduto separatamente), nel caso in cui non si desideri alimentare il Q25 tramite il collegamento USB.
2. **PORTA USB –** Inserire un cavo standard USB a livello di questa presa e nella porta USB del computer. La porta USB del computer fornirà l'alimentazione al Q25. Questo collegamento serve per inviare e ricevere dati MIDI da e verso il computer e può anche essere utilizzato per l'invio di dati MIDI dal computer ad un dispositivo collegato alla porta MIDI OUT del Q25.
3. **USCITA MIDI –** Servirsi di un cavo MIDI a cinque poli (venduto separatamente) per collegare questo jack all'ingresso MIDI di un dispositivo esterno.
4. **INGRESSO PER PEDALE SUSTAIN –** Collegare un pedale sustain TS da 1/4" (venduto separatamente) a questo ingresso.

PANORAMICA PANNELLO SUPERIORE



1. **TASTIERA –** La tastiera funziona come una normale tastiera elettronica durante le esibizioni, ma può anche essere utilizzata per regolare le impostazioni MIDI e per inviare messaggi MIDI. Le etichette al di sopra dei tasti ne indicano le funzioni. I tasti numerati consentono di inserire valori per le configurazioni. Premere il tasto Cancel o Enter rispettivamente per annullare o confermare una scelta. Per maggiori informazioni, si veda il paragrafo SELEZIONE E MODIFICA DEI COMANDI MIDI.
2. **ROTELLA DI BEND DEL PITCH –** Trasmette le informazioni relative al bend del pitch MIDI per alzare o abbassare temporaneamente il pitch di una nota.
3. **ROTELLA DI MODULAZIONE –** Questa rotella può essere utilizzata per trasmettere dati continui del controller (CC #1 o profondità di modulazione).
4. **CURSORE DI INSERIMENTO DATI** – Questo cursore permette di inviare messaggi MIDI per i parametri attualmente selezionati dalla TASTIERA (ad es., REVERB DEPTH, VOLUME, ecc.). Si può selezionare un parametro premendo MIDI / SELECT e quindi il tasto corrispondente sulla TASTIERA.
5. **MIDI / SELECT –** Premendo questo tasto è possibile regolare le impostazioni MIDI e inviare messaggi MIDI premendo i tasti etichettati sulla TASTIERA. Per maggiori informazioni, si veda il paragrafo SELEZIONE E MODIFICA DEI COMANDI MIDI.
6. **OCTAVE UP / DOWN –** Questi tasti possono essere utilizzati per alzare o abbassare la gamma della tastiera.

SELEZIONE E MODIFICA DEI COMANDI MIDI

Il tasto MIDI / SELECT e i tasti della TASTIERA consentono di regolare i parametri MIDI e di inviare messaggi MIDI precisi e informazioni in maniera semplice e rapida:

1. **Premere MIDI / SELECT.**
2. **Premere il tasto della TASTIERA la cui impostazione MIDI (stampata sul tasto stesso) si desidera modificare o inviare.**
3. **Inserire un valore con i tasti numerati sulla TASTIERA.**
4. **Premere i tasti Enter o Cancel sulla TASTIERA per confermare o annullare la scelta, rispettivamente.**
5. **Premere MIDI / SELECT se il pulsante è ancora acceso.**

VELOCITÀ – Premere questo tasto per fare in modo che il CURSORE DI INSERIMENTO DATI regoli la velocità delle note. Aumentando questo valore aumenta il valore di velocità MIDI generato da una leggera pressione dei tasti.

AFTERTOUCH – Premere questo tasto per fare in modo che il CURSORE DI INSERIMENTO DATI invii informazioni Aftertouch (talvolta anche denominato Pressione di canale).

REVERB DEPTH (profondità reverb) – Premere questo tasto per impostare il CURSORE DI INSERIMENTO DATI su CC #91 (Reverb Send Level). Il valore predefinito di questa impostazione è 64.

VOLUME – Premere questo tasto per impostare il CURSORE DI INSERIMENTO DATI su CC #7 (Volume di canale).

CC DATA (Control Change Data) **–** Il valore da inviare. Con la pressione del tasto Enter invia un messaggio MIDI.

b / # – Abbassa o alza (rispettivamente) il pitch dell'intera TASTIERA di un semitono.

RESET – Premere questo tasto seguito dal tasto Enter per inviare un messaggio di azzeramento a tutti i controller e far tornare il Q25 alla sua configurazione originale.

Nota bene: questo tasto non azzera la gamma corrente di ottave o la trasposizione della TASTIERA.

CC NO (Control Change Number) **–** Imposta il numero di CC MIDI da inviare. (Non verranno inviati messaggi fino a quando non viene premuto il tasto Enter al momento di selezionare i dati CC.)

PROGRAM – Imposta il numero di MIDI Program Change.

Nota bene: *dopo aver inserito un numero di CC, questo verrà memorizzato dal Q25. Tuttavia, se si desidera inviare un CC o un messaggio Program Change occorre inserire il valore servendosi dei tasti numerati sulla TASTIERA ogni volta (anche se il valore desiderato compare sullo schermo LCD) prima di premere Enter.*

RISOLUZIONE DI PROBLEMI

PROBLEMA	CAUSA	SOLUZIONE
Il display non si accende.	Mancanza di corrente.	Assicurarsi che il Q25 sia collegato al computer e che il computer sia acceso.
		Se si utilizza un adattatore di alimentazione, assicurarsi che sia collegato ad una presa elettrica funzionante.
Nessun suono dal dispositivo target.	Il Q25 non è stato collegato correttamente.	Verificare il collegamento USB del computer per assicurarsi che il Q25 sia riconosciuto. Se necessario, ricollegare l'apparecchio e riavviare il computer.
		Se si controlla un modulo hardware esterno, assicurarsi che il cavo MIDI sia collegato dal Q25 alla porta MIDI IN del dispositivo.
	Il Q25 è stato collegato in seguito al lancio dell'applicazione software.	Riavviare l'applicazione software con il controller inserito.
	Il problema è causato dall'uso di un hub USB.	Scollegare il Q25 dall'hub USB e collegarlo direttamente al computer.
	L'applicazione software non è impostata per ricevere dati MIDI dal Q25.	Assicurarsi che il Q25 o dispositivo MIDI USB sia elencato come sorgente MIDI attiva nell'applicazione. Solitamente, è possibile accedere alle impostazioni MIDI tramite il menu Preferences dell'applicazione.
	Il canale MIDI del Q25 non è lo stesso del canale MIDI in ingresso dell'applicazione.	Assicurarsi che il Q25 invii informazioni MIDI sul canale che il dispositivo target si aspetta.
Le note sono sostenute in maniera continua	Il pedale sustain è stato collegato dopo che il Q25 è stato acceso.	Spegnere l'apparecchio, attendere alcuni istanti, quindi riaccenderlo.
	Note bloccate per via di dati MIDI incompleti.	Spegnere l'apparecchio, attendere alcuni istanti, quindi riaccenderlo.
Il pedale sustain funziona al contrario.	Il pedale sustain è stato collegato dopo che l'alimentazione è stata accesa.	Con il pedale collegato, spegnere l'apparecchio, attendere alcuni istanti, quindi riaccenderlo.

SPECIFICHE TECNICHE

ALIMENTAZIONE: USB, 9V CC, 500mA, positiva centrale, diametro 5,46mm (venduto separatamente)
TASTIERA: 25 tasti
ACCESSORI: guida rapida, cavo USB
USCITE MIDI: 1 jack a 5 poli
USB: 1 connettore slave (MIDI su USB)

http://www.alesis.com/q25

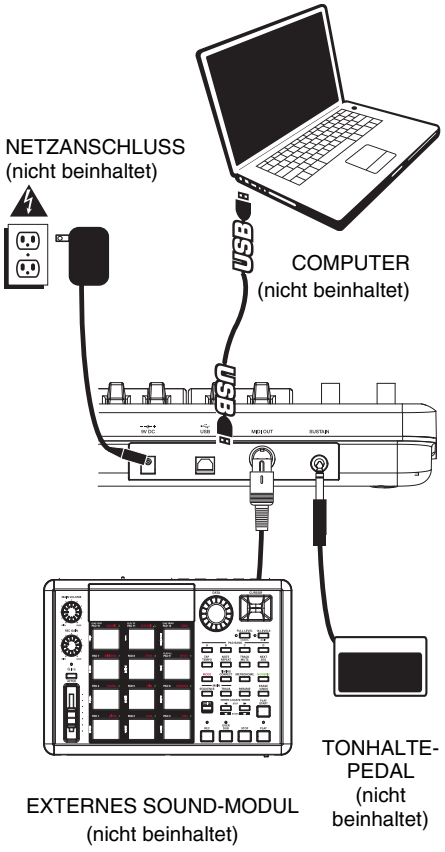
EINLEITUNG

Diese Schnellanleitung enthält eine kurze Übersicht der Bedienelemente und Funktionen des Q25. Hier finden Sie auch Anweisungen, wie das Q25 angeschlossen und bedient wird. Wir wünschen Ihnen damit viel Freude!

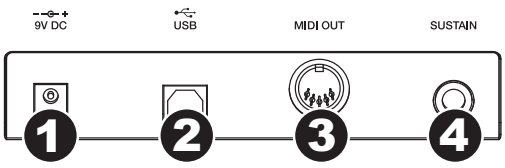
ANSCHLUSSDIAGRAMM

Legen Sie beim Anschluss des Q25 bitte die folgende Situation zugrunde.

1. Das Q25 mit einem USB-Kabel an den Computer anschließen. Das Gerät wird über den USB-Anschluss mit Strom versorgt. Anderenfalls, sollte der Computer nicht Bestandteil des Aufbaus sein oder falls das Q25 über eine externe Stromquelle gespeist werden soll, bitte einen 9V-Gleichstromadapter (center-positive) mit 500mA, 5,46mm Durchmesser (separat erhältlich) einsetzen.
2. Falls ein externes Sound-Modul verwendet werden soll, das Q25 mit einem fünfpoligen MIDI-Kabel über den Ausgang KEYBOARD MIDI OUT mit dem Eingang MIDI IN des externen Geräts verbinden.

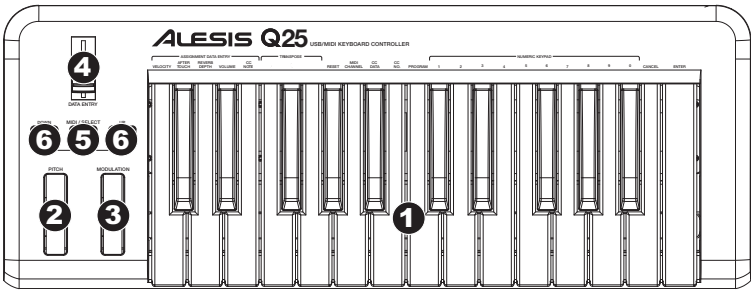


FUNKTIONSELEMENTE GERÄTERÜCKSEITE



1. **NETZANSCHLUSS FÜR GLEICHSTROMADAPTER** – Zum Anschluss eines 9V-Gleichstromadapters (center-positive) mit 500mA, 5,46mm Durchmesser (separat erhältlich), falls das Q25 nicht über den USB-Anschluss mit Strom gespeist werden soll.
2. **USB-ANSCHLUSS –** Anschluss eines herkömmlichen USB-Kabels, dessen anderes Ende in einen USB-Anschluss am Computer angeschlossen wird. Der USB-Anschluss des Computers versorgt das Q25 mit Strom. Diese Verbindung dient zum Senden und Empfangen von MIDI-Daten zwischen Dem Computer und dem Q25 und kann auch dazu verwendet werden, MIDI-Daten vom Computer an ein Gerät, welches an den Ausgang MIDI OUT des Q25 angeschlossen ist, zu senden.
3. **MIDI AUS –** Diesen Anschluss über ein fünfpoliges MIDI-Kabel (separat erhältlich) mit dem Anschluss MIDI IN eines externen Geräts verbinden.
4. **EINGANG TONHALTEPEDAL –** Dient zum Anschluss eines 6,35 mm TS-Tonhaltepedals (separat erhältlich).

FUNKTIONSELEMENTE GERÄTEOBERSEITE



1. **KEYBOARD –** Das KEYBOARD fungiert als normale, elektronische Pianotastatur während der Aufführung, kann aber auch zum Einstellen der MIDI-Optionen und zum Senden von MIDI-Meldungen verwendet werden. Die oberhalb der Tasten befindlichen Bezeichnungen zeigen deren Funktion an. Die numerierten Tasten dienen zur Eingabe von Parameterwerten. Mit den Tasten Cancel oder Enter werden die gewählten Werte jeweils gelöscht oder bestätigt. Weitere Informationen finden sich im Abschnitt AUSWAHL & EDITIEREN VON MIDI-BEFEHLEN.
2. **TONHÖHENVERBIEGUNGSRAD –** Sendet Mitteilungen zur Tonhöhenverbiegung im MIDI-Format um die Tonhöhe einer Note zeitweise anzuheben oder abzusenken.
3. **MODULATIONSRAD –** Mit diesem Rad können fortwährende Steuerungsdaten (CC #1 oder Modulationstiefe) übermittelt werden.
4. **DATENEINGABEREGLER** – Mit diesem Regler können MIDI-Mitteilungen für den augenblicklich auf der Tastatur gewählten Parameter übermittelt werden (z. B. REVERB DEPTH, LAUTSTÄRKE usw.). Der Parameter kann durch Drücken der Taste MIDI / SELECT und der jeweiligen Taste auf der TASTATUR.
5. **MIDI / SELECT –** Durch Drücken dieser Taste können die MIDI-Einstellungen verändert und MIDI-Meldungen über die gekennzeichneten Tasten des KEYBOARDS gesendet werden. Weitere Informationen finden sich im Abschnitt AUSWAHL & EDITIEREN VON MIDI-BEFEHLEN.
6. **OKTAVE AUF / AB –** Mit diesen Tasten kann der Oktavenbereich der Tastatur nach oben oder unten verlagert werden.

AUSWAHL & EDITIEREN VON MIDI-BEFEHLEN

Die MIDI / SELECT-Taste und die Tasten des KEYBOARDS können zur Einstellung der MIDI-Parameter und zum leichten und schnellen Senden von MIDI-Meldungen und –Informationen verwendet werden:

1. **MIDI / SELECT-Taste drücken.**
2. **Auf dem KEYBOARD die Taste drücken, deren MIDI-Einstellung (ersichtlich oberhalb der Taste) geändert oder übermittelt werden soll.**
3. **Mit den numerierten Tasten des KEYBOARDS einen Wert eingeben.**
4. **Mit Enter bestätigen oder mit Cancel löschen.**
5. **MIDI / SELECT-Taste drücken, wenn die Taste leuchtet noch.**

VELOCITY – Mit dieser Taste stellt man den DATENEINGABEREGLER darauf ein, die Notengeschwindigkeit einzustellen. Wird dieser Wert erhöht, steigt der durch ein leichtes Spielen der Taste erzeugte MIDI-Geschwindigkeitswert.

AFTERTOUCH – Mit dieser Taste stellt man den DATENEINGABEREGLER darauf ein, Aftertouch-Informationen (auch manchmal als "Channel Pressure" bezeichnet) zu senden.

REVERB DEPTH – Mit dieser Taste wird der DATENEINGABEREGLER auf CC #91 (Reverb-Sendepegel) eingestellt. Der ab Werk eingestellte Wert beträgt 64.

VOLUME – Mit dieser Taste wird der DATENEINGABEREGLER auf CC #7 (Kanal-Lautstärke) eingestellt.

CC DATA (Control Change Data) **–** Zum Einstellen des zu sendenden Werts. Bei Drücken der Enter-Taste wird eine MIDI-Mitteilung gesendet.

b / # – Diese Taste dient zum Absenken/Anheben des Tonpegels des gesamten KEYBOARDS um eine halbe Note.

RESET – Durch Drücken dieser Taste und einer Bestätigung mit ENTER wird ein Rücksetzbefehl an alle Steuerungseinheiten gesendet und das Q25 stellt wieder die ab Werk eingegebenen Werte ein.
Hinweis: Diese Taste ändert den augenblicklichen Oktavenbereich oder die Notenverlagerung des KEYBOARDS nicht.

CC NO. (Control Change Number) **–** Dient zur Einstellung des MIDI-CC# Wertes, der gesendet werden soll. (Bei der Auswahl der CC-Werte wird eine Mitteilung wird erst dann gesendet, wenn die Enter-Taste gedrückt wird).

PROGRAM – Diese Taste dient zum Einstellen der MIDI-Programmwechselnummer.

Hinweis: *Nach Eingabe einer CC#-Nummer, wird diese vom Q25 gespeichert. Falls jedoch eine CC-Nummer oder eine Programmwechselmitteilung gesendet werden soll, muss der Wert mit den numerierten Tasten über das KEYBOARD jedes Mal, bevor die Enter-Taste gedrückt wird, eingegeben werden (selbst, wenn der gewünschte Wert in der LCD-Anzeige angezeigt wird.*

FEHLERBEHEBUNG

PROBLEM	URSACHE	LÖSUNG
Anzeige ist nicht beleuchtet.	Kein Strom.	Überprüfen, ob das Q25 an den Computer angeschlossen ist und dieser eingeschaltet ist.
		Bei Verwendung eines Netzadapters überprüfen, der Adapter in eine unter Strom stehende Steckdose gesteckt wurde.
Kein Sound vom Zielgerät.	Q25 ist nicht vorschriftsmäßig angeschlossen.	USB-Verbindung des Computers überprüfen und sicherstellen, dass das Q25 erkannt wird. Falls notwendig, Verbindung erneut aufbauen und Computer erneut starten.
		Wird über die Verbindung ein externes Gerät gesteuert, sicherstellen, dass das MIDI-Kabel vom Q25 an den MIDI-EINGANG des Geräts angeschlosssen ist.
	Q25 wurde angeschlossen, nachdem das Programm gestartet wurde.	Programm mit dem angeschlossenen Steuergerät erneut starten.
	Problem wird von einem USB-Hub verursacht.	Q25 aus dem USB-Hub ausstecken und direkt an den Computer anschließen.
	Programmeinstellungen gestatten keinen Empfang von MIDI-Daten vom Q25.	Überprüfen, dass das Q25 oder ein USB MIDI-Gerät als aktive MIDI-Quelle im Programm erscheinen. Auf die MIDI-Einstellungen kann für gewöhnlich über das Einstellungs- oder Optionsmenü des Programms zugegriffen werden.
	MIDI-Kanal am Q25 und MIDI-Eingangskanal des Programms stimmen nicht überein.	Überprüfen, dass das Q25 MIDI-Informationen auf dem Kanal sendet, der vom Zielgerät erwartet wird.
Noten werden ständig gehalten.	Tonhaltepedal wurde nach Einschalten des Q25 angeschlossen.	Gerät ausschalten, einen Moment warten und dann wieder einschalten.
	Noten bleiben aufgrund von unvollständigen MIDI-Daten hängen.	Gerät ausschalten, einen Moment warten und dann wieder einschalten.
Tonhaltepedal funktioniert gegensätzlich.	Tonhaltepedal wurde nach Einschalten des Q25 angeschlossen.	Mit dem Pedal an das Gerät angeschlossen dieses ausschalten, einen Moment warten und dann wieder einschalten.

SPEZIFIKATIONEN

NETZ: USB, 9V DC, 500mA, center-positiv, 5,46mm Durchmesser (separat erhältlich)
KEYBOARD: 25 Tasten
ZUBEHÖR: Schnellbedienungsanleitung, USB-Kabel
MIDI AUSGÄNGE: 1 5-poliger Anschluss
USB: 1 Überbrückungsanschluss (MIDI über USB)

http://www.alesis.com/q25

MIDI IMPLEMENTATION CHART

	Transmit/Export	Recognize/Import	Remarks
1. Basic Information			
MIDI channels	1-16	No	Default = 1
Note numbers	12-108	No	With Octave +/- buttons
Program change	1-128	No	
Bank Select response	Yes	No	0-127
Modes supported: Mode 1: Omni-On, Poly Mode 2: Omni-On, Mono Mode 3: Omni-Off, Poly Mode 4: Omni-Off, Mono Multi Mode	No No Yes No No	No No No No No	
Note-On Velocity	Yes	No	
Note-Off Velocity	No	No	
Channel Aftertouch	Yes	No	Via data slider, when assigned
Poly (Key) Aftertouch	No	No	
Pitch Bend	Yes	No	
Active Sensing	No	No	
System Reset	Yes	No	
Tune Request	No	No	
Universal System Exclusive	No	No	
Manufacturer or Non-Commercial System Exclusive	No	No	
NRPNs	No	No	
RPNs	No	No	
2. MIDI Timing and Synchronization			
MIDI Clock	No	No	
Song Position Pointer	No	No	
Song Select	No	No	
Start Continue Stop	No No No	No No No	
MIDI Time Code	No	No	
MIDI Machine Control	No	No	
MIDI Show Control	No	No	

www.alesis.com