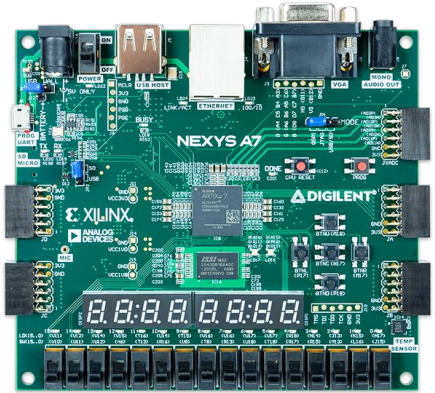
FPGA Synthesizer

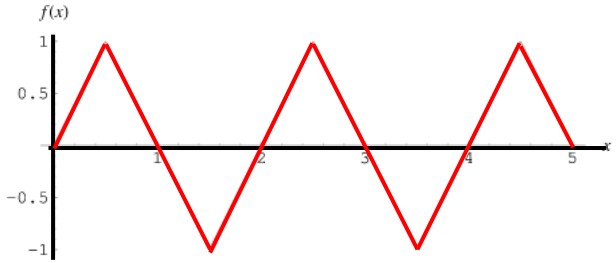
Nihal Hossain, Matthew Petrin, Ahmed Abuharthieh

What are we doing?

- The purpose of this project is to utilize some fundamental digital logic and VHDL concepts learned in this course to develop an FPGA Synthesizer.
- The board takes inputs from the user via a PMOD Keypad and plays back a signed 16-bit triangle wave in the corresponding pitch

  (1 = C5, 2 = C#5, 3 = D5, etc.)

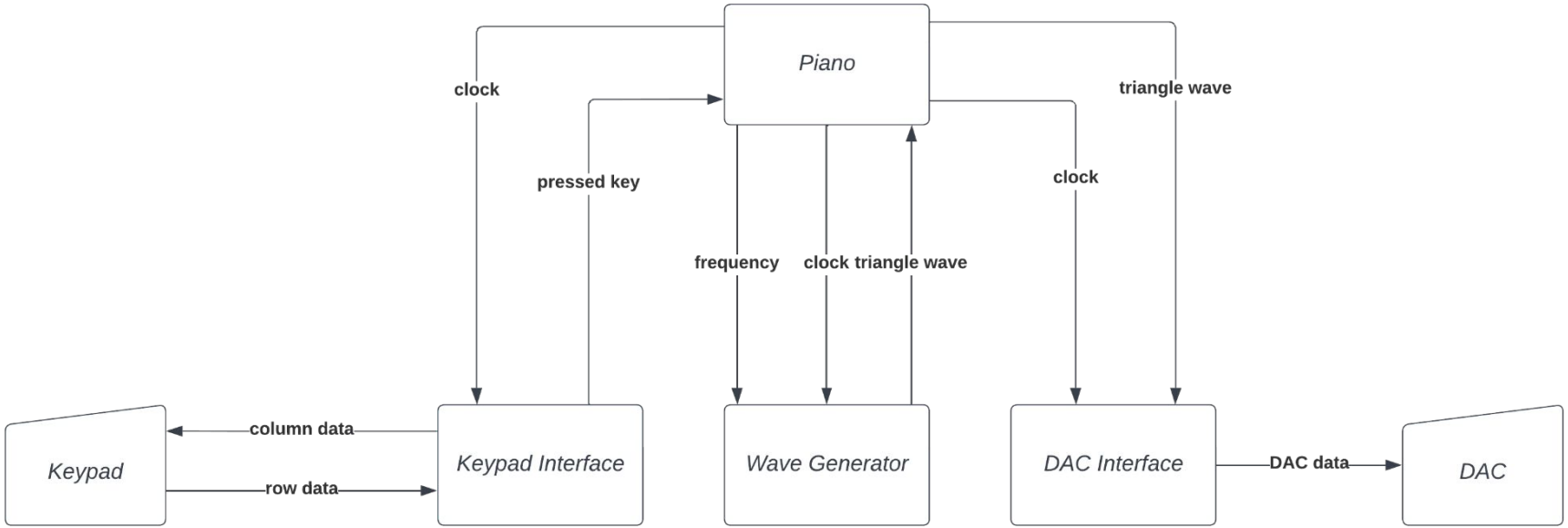




What is a Synthesizer?

A synthesizer is an electronic instrument that uses digital or audio processing to produce audio. The use of this instrument is based on its ability to artificially reproduce other instruments sounds and produce it on its own. Synthesizers are able to produce sounds by stacking different wave types on one another. Then low pass and high pass filters are utilized to extract the high and low frequencies and play those sounds.

Block Diagram



VHDL Architecture

- Playable notes defined as constant unsigned vectors
- Interfaces and wave generator as components
- Internal signals to be assigned to component ports

```
-- Piano architecture definition ----------------------------------------------------------------
architecture Behavioral of piano is
    -- Note constants in 14 bits ---------------------------------------------------------------
    constant NOTE_NULL : unsigned (13 downto 0) := to_unsigned (0, 14);   -- 0 Hz wave
    constant NOTE_01 : unsigned (13 downto 0) := to_unsigned (351, 14); -- C
    constant NOTE_02 : unsigned (13 downto 0) := to_unsigned (372, 14); -- C#
    constant NOTE_03 : unsigned (13 downto 0) := to_unsigned (394, 14); -- D
    constant NOTE_04 : unsigned (13 downto 0) := to_unsigned (418, 14); -- Eb
    constant NOTE_05 : unsigned (13 downto 0) := to_unsigned (442, 14); -- E
    constant NOTE_06 : unsigned (13 downto 0) := to_unsigned (469, 14); -- F
    constant NOTE_07 : unsigned (13 downto 0) := to_unsigned (497, 14); -- F#
    constant NOTE_08 : unsigned (13 downto 0) := to_unsigned (526, 14); -- G
    constant NOTE_09 : unsigned (13 downto 0) := to_unsigned (557, 14); -- G#
    constant NOTE_10 : unsigned (13 downto 0) := to_unsigned (591, 14); -- A
    constant NOTE_11 : unsigned (13 downto 0) := to_unsigned (626, 14); -- Bb
    constant NOTE_12 : unsigned (13 downto 0) := to_unsigned (663, 14); -- B
    constant NOTE_13 : unsigned (13 downto 0) := to_unsigned (702, 14); -- B
    -- http://www.sengpielaudio.com/calculator-notenames.htm (units of 0.745 Hz)
```

```
    -- Keypad component definition ------------------------------------------------------------
    component kpd_interface is
        port (
            clk : in std_logic;
            row : in std_logic_vector(4 downto 1);
            col : out std_logic_vector(4 downto 1);
            val : out std_logic_vector(3 downto 0);
            hit : out std_logic
        );
    end component;

    -- Tone component definition --------------------------------------------------------------
    component tone_generator is
        port (
            clk : in std_logic;
            note: in unsigned (13 downto 0);
            data: out signed (15 downto 0)
        );
    end component;

    -- DAC component definition ---------------------------------------------------------------
    component dac_interface is
        port(
            clk : in std_logic;
            l_load : in std_logic;
            r_load : in std_logic;
            l_data : in signed (15 downto 0);
            r_data : in signed (15 downto 0);
            data : out std_logic
        );
    end component;
```

```
    -- Signals passed to keypad ---------------------------------------------------------------
    signal kpd_cnt : std_logic_vector(20 downto 0);
    signal kpd_clk : std_logic;
    signal kpd_hit : std_logic;
    signal kpd_val : std_logic_vector (3 downto 0);

    -- Signals passed to tone -----------------------------------------------------------------
    signal ton_clk : std_logic;
    signal ton_note : unsigned (13 downto 0);
    signal ton_wave : unsigned (2 downto 0);

    -- Signals passed to DAC ------------------------------------------------------------------
    signal dac_cnt : unsigned (19 downto 0) := (others => '0');
    signal dac_l_load : std_logic;
    signal dac_r_load : std_logic;
    signal dac_data : signed (15 downto 0);
    signal dac_clk : std_logic;
```

Architecture

Keypad Interface:

- Provides input that holds column low
- Button press holds a row low
- Board reports low button given column

DAC Interface:

- Loads data into DAC serially
- Receives clock signals for master clock and left-right clock
- DAC determines serial clock automatically

Tone Generator

- Receives frequency at 1.33x desired frequency
- Outputs signed triangle wave approximating sine wave

VHDL Models

Behavioral:

- Checks if a key is pressed and which key is pressed
- Assigns note signal the correct constant frequency
- Assigns 0 Hz triangle wave if no key is pressed

```
151     -- Keypad switch statement ------------------------------------------------------------------------
152     kpd_proc : process (kpd_hit, kpd_val, clk_50MHz)
153     begin
154         if kpd_hit = '1' then
155             case kpd_val is
156                 when X"1" =>
157                 ton_note <= NOTE_01;
158                 when X"2" =>
159                 ton_note <= NOTE_02;
160                 when X"3" =>
161                 ton_note <= NOTE_03;
162                 when X"A" =>
163                 ton_note <= NOTE_04;
164                 when X"4" =>
165                 ton_note <= NOTE_05;
166                 when X"5" =>
167                 ton_note <= NOTE_06;
168                 when X"6" =>
169                 ton_note <= NOTE_07;
170                 when X"B" =>
171                 ton_note <= NOTE_08;
172                 when X"7" =>
173                 ton_note <= NOTE_09;
174                 when X"8" =>
175                 ton_note <= NOTE_10;
176                 when X"9" =>
177                 ton_note <= NOTE_11;
178                 when X"C" =>
179                 ton_note <= NOTE_12;
180                 when X"0" =>
181                 ton_note <= NOTE_13;
182                 when others =>
183                 ton_note <= NOTE_NULL;
184             end case;
185 
186             else
187                 ton_note <= NOTE_NULL;
188         end if;
189     end process;
190 end Behavioral;
```

Clock Problems

- We use values for the DAC interface and wave generator calculated for a 50 MHz clock
- This has no effect on the DAC, because the DAC operates on clock ratios (both doubled)
- However, our notes are produced one octave higher than expected based on inputs

LRCK (kHz)	MCLK (MHz)									
	64x	96x	128x	192x	256x	384x	512x	768x	1024x	1152x
32	-	-	-	-	8.1920	12.2880	-	-	32.7680	36.8640
44.1	-	-	-	-	11.2896	16.9344	22.5792	33.8680	45.1580	-
48	-	-	-	-	12.2880	18.4320	24.5760	36.8640	49.1520	-
64	-	-	8.1920	12.2880	-	-	32.7680	49.1520	-	-
88.2	-	-	11.2896	16.9344	22.5792	33.8680	-	-	-	-
96	-	-	12.2880	18.4320	24.5760	36.8640	-	-	-	-
128	8.1920	12.2880	-	-	32.7680	49.1520	-	-	-	-
176.4	11.2896	16.9344	22.5792	33.8680	-	-	-	-	-	-
192	12.2880	18.4320	24.5760	36.8640	-	-	-	-	-	-
Mode	QSM				DSM		SSM			

Table 1. Common Clock Frequencies

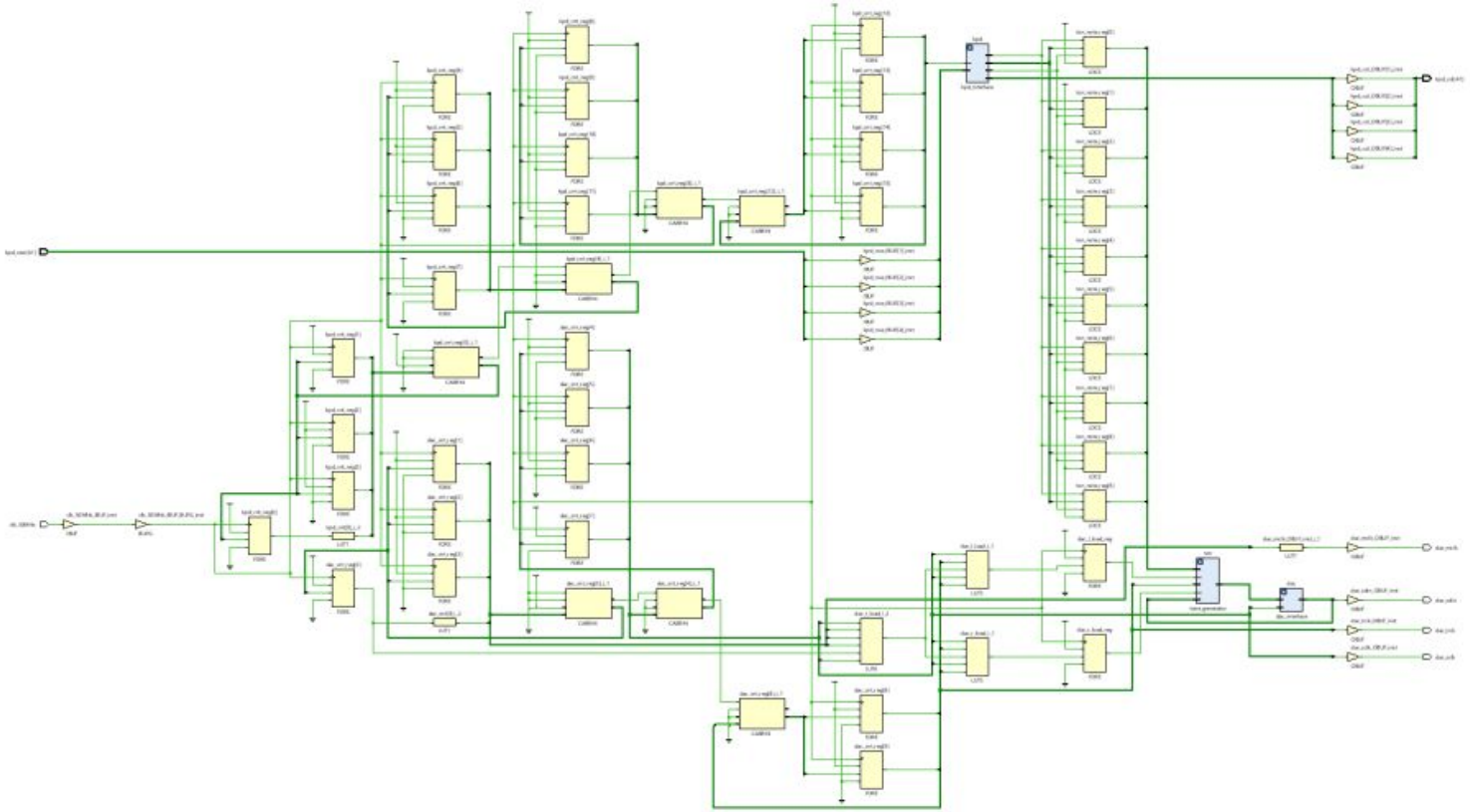
```
-- Clocks created by counter signals
ton_clk <= dac_cnt(9);
dac_clk <= dac_cnt(4);
kpd_clk <= kpd_cnt(15);

-- DAC clock assignments ------------
dac_mclk <= NOT dac_cnt(1);
dac_lrck <= ton_clk;
dac_sclk <= dac_clk;
```

VHDL Component Reuse

Hardware clock is used to increment both the DAC and keypad counters

```
107             dac_cnt <= dac_cnt + 1;
108             kpd_cnt <= kpd_cnt + 1;
109         end if;
110     end process;
111
112     -- Clocks created by counter signals ------------------------------------------------------------
113     ton_clk <= dac_cnt(9);
114     dac_clk <= dac_cnt(4);
115     kpd_clk <= kpd_cnt(15);
116
117     -- DAC clock assignments ------------------------------------------------------------------------
118     dac_mclk <= NOT dac_cnt(1);
119     dac_lrck <= ton_clk;
120     dac_sclk <= dac_clk;
```

VHDL Digital Circuits



State Machine

- Moore State machine: The device's operation ultimately depends on the internal clock. The input alone is not enough to affect the output.
- If the clock is stopped, the machine ceases to function

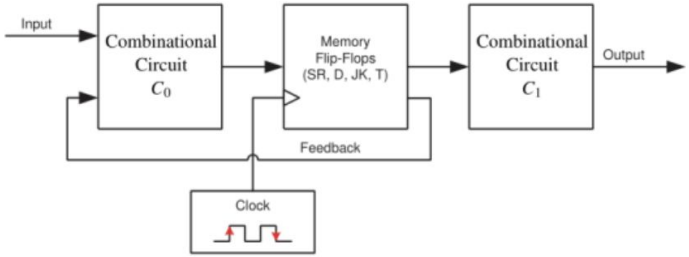


Figure 2.1: Moore state machines.

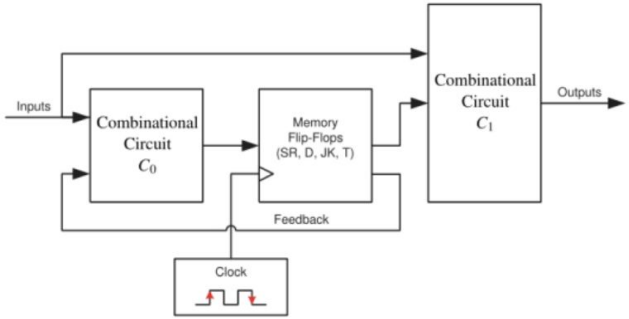
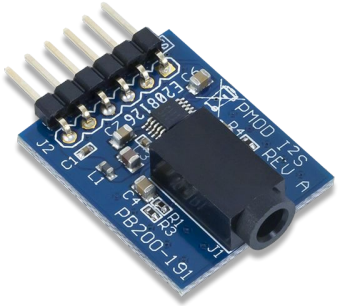


Figure 2.2: Mealy state machines.

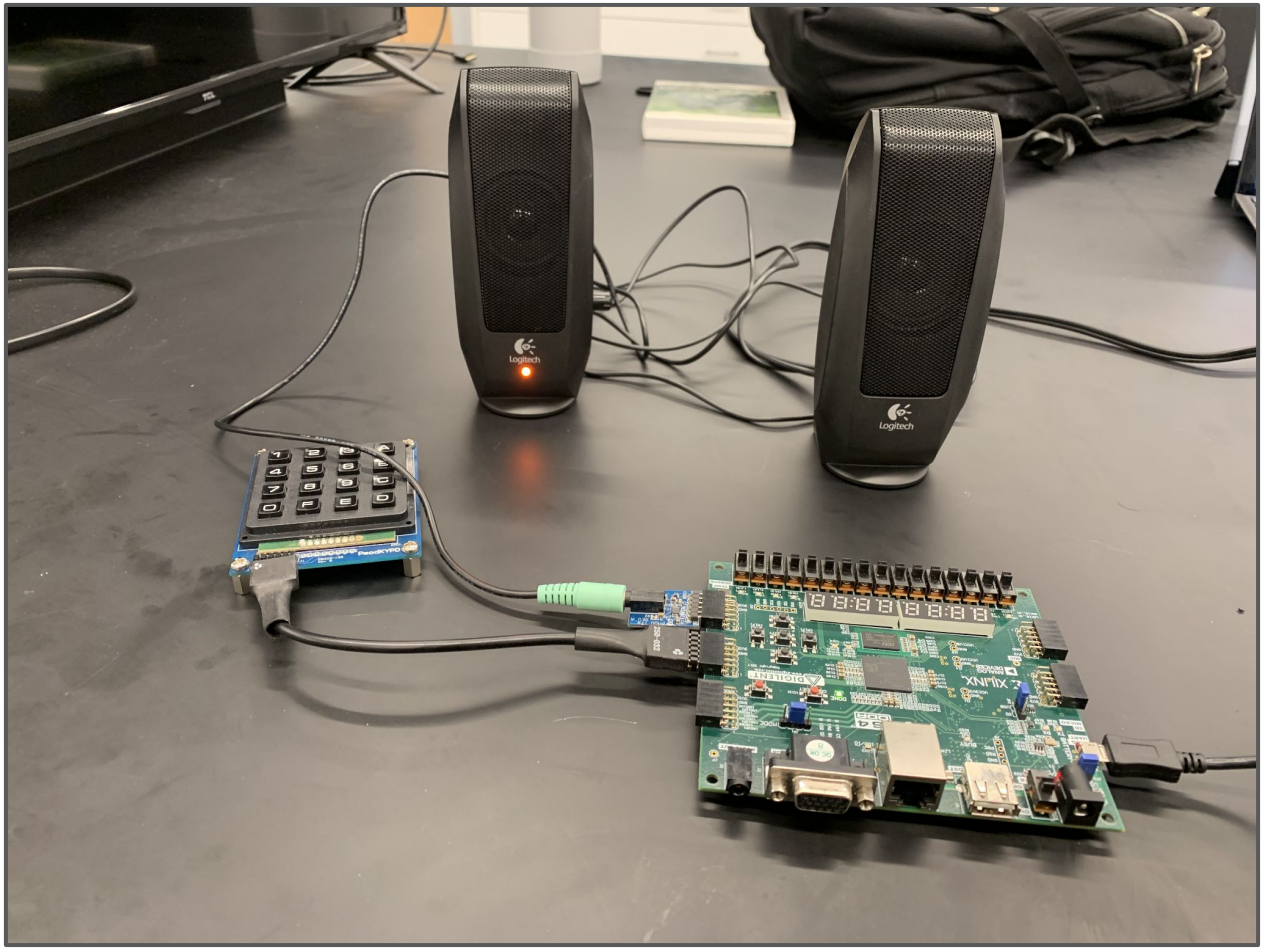
Off Board Devices

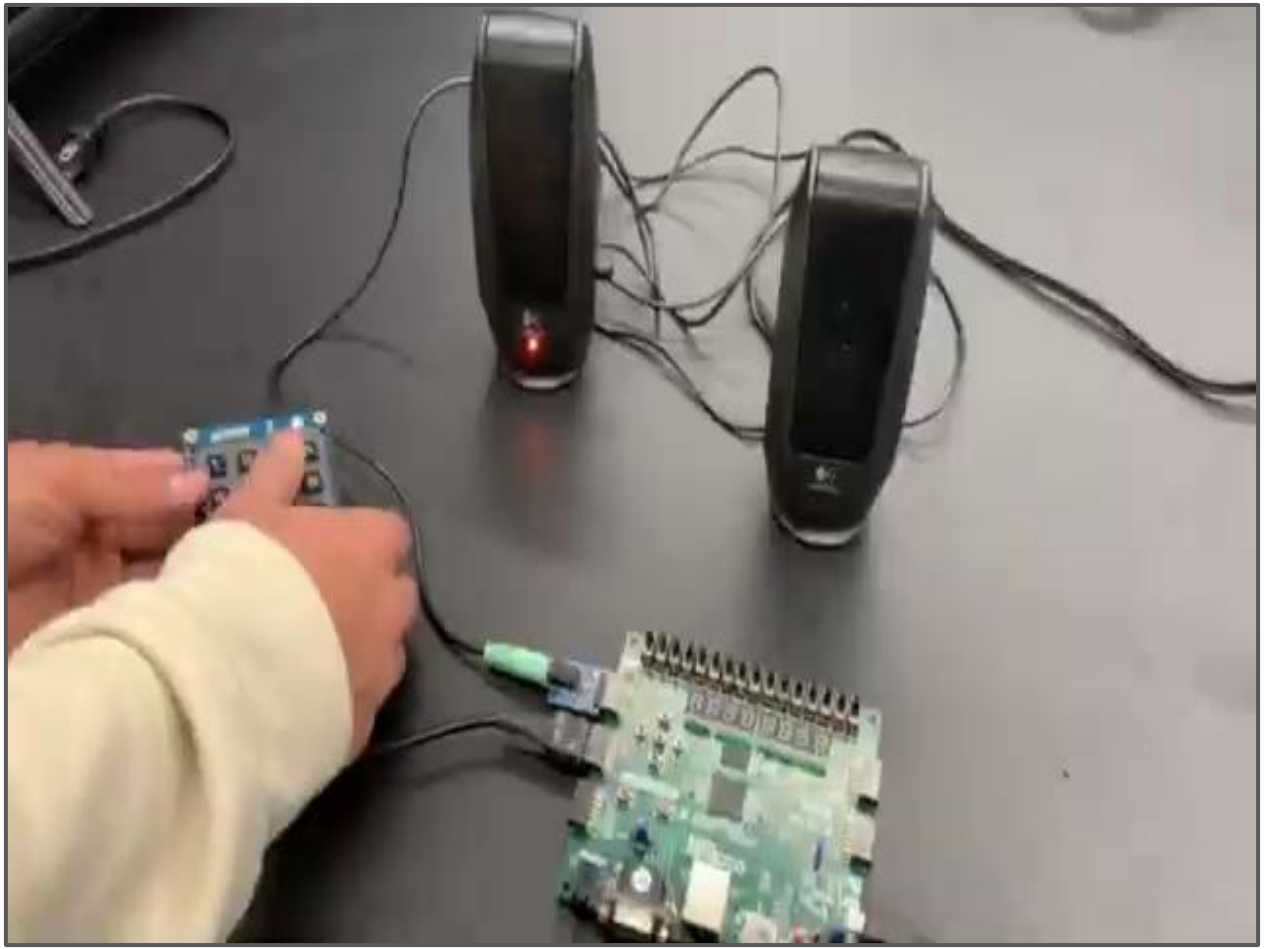


- PmodI2S DAC
- PmodKYPD Keypad

```
# Clock signal
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk_50MHz}];
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports {clk_50MHz}]

# PMOD Header JA
set_property -dict { PACKAGE_PIN C17 IOSTANDARD LVCMOS33 } [get_ports { kpd_col[4] }]; #IO_L20N_T3_A19_15 Sch=ja[1]
set_property -dict { PACKAGE_PIN D18 IOSTANDARD LVCMOS33 } [get_ports { kpd_col[3] }]; #IO_L21N_T3_DQS_A18_15 Sch=ja[2]
set_property -dict { PACKAGE_PIN E18 IOSTANDARD LVCMOS33 } [get_ports { kpd_col[2] }]; #IO_L21P_T3_DQS_15 Sch=ja[3]
set_property -dict { PACKAGE_PIN G17 IOSTANDARD LVCMOS33 } [get_ports { kpd_col[1] }]; #IO_L18N_T2_A23_15 Sch=ja[4]
set_property -dict { PACKAGE_PIN D17 IOSTANDARD LVCMOS33 } [get_ports { kpd_row[4] }]; #IO_L16N_T2_A27_15 Sch=ja[7]
set_property -dict { PACKAGE_PIN E17 IOSTANDARD LVCMOS33 } [get_ports { kpd_row[3] }]; #IO_L16P_T2_A28_15 Sch=ja[8]
set_property -dict { PACKAGE_PIN F18 IOSTANDARD LVCMOS33 } [get_ports { kpd_row[2] }]; #IO_L22N_T3_A16_15 Sch=ja[9]
set_property -dict { PACKAGE_PIN G18 IOSTANDARD LVCMOS33 } [get_ports { kpd_row[1] }]; #IO_L22P_T3_A17_15 Sch=ja[10]

# PMOD Header JB
set_property -dict { PACKAGE_PIN F16 IOSTANDARD LVCMOS33 } [get_ports { dac_lrck }]; #IO_L21N_T3_DQS_A18_15 Sch=ja[2]
set_property -dict { PACKAGE_PIN G16 IOSTANDARD LVCMOS33 } [get_ports { dac_sclk }]; #IO_L21P_T3_DQS_15 Sch=ja[3]
set_property -dict { PACKAGE_PIN H14 IOSTANDARD LVCMOS33 } [get_ports { dac_sdin }]; #IO_L18N_T2_A23_15 Sch=ja[4]
set property -dict { PACKAGE_PIN D14 IOSTANDARD LVCMOS33 } [get ports { dac_mclk }]; #IO L20N T3 A19 15 Sch=ja[1]
```

Testing and Demonstration


Logitech
Logitech
XILINX



Potential Extensions/Improvements

- The synth could be expanded to include multiple types of sound waves, allowing for the creation of a variety of different sounds
- An additional “Octave Position” state could be added to make the device capable of playing notes in a wider range
- A real midi keyboard could potentially be used instead of the PMOD keypad to allow for more natural playing