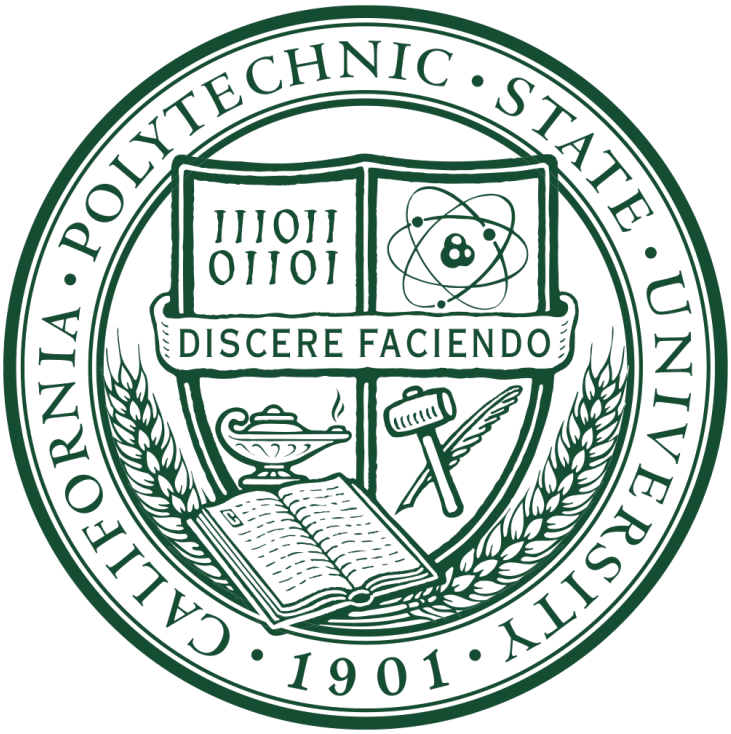
CS 133: Digital Design

User-Input Temperature Converter Designed with VIVADO on the BASYS3 Board

By: Roee Landesman



Introduction

The assigned goal of the Midterm Project was to fully design, simulate, and implement a circuit per the specifications provided by the Instructor. The overall task was to create a circuit which would accept a 3-4 bit input from a user and –again according to the user's selection—output the conversion into either Celsius, Fahrenheit, or Kelvin. The required components of this project include one of each of the following: Multiplexer (MUX/Selector), Register (JK Flip Flop), Decoder, Ripple Carry Adder (RCA), and a circuit designed by the student. This report will detail each of the components mentioned, as well as the logic of the main file. Evidence of all VHDL code will be put in the appendix.

Usability/ Physical Representation

The final circuit has six user inputs and four digital outputs, all of which are represented using LEDs and Switches on the BASYS3 board. Four of the inputs are dedicated to the user's initial temperature input, and are conveniently placed as the (four) far right switches. The far-left switch, which is represented as *celc* in VHDL, is used by the user if he/she wants to bypass all operations and return the input number. One switch to the right of *celc* is the *sel* switch, which is used by the user to determine the output between Fahrenheit and Kelvin; If *sel* is on (1), the output will be in Fahrenheit, and if it is off (0) the output will be Kelvin. All components run real time without added delays.

Components

1. Multiplexer/Selector
   The MUX that is implemented in my design has three inputs and one output. Two of the outputs, represented by *a* and *b* in VHDL, represent the input bits that are to be selected from. The third input, shown as *sel*, is used to determine the output of the component. This is a standard selector which outputs *b* if the selector is 1, and *a* if the selector is off. In this project's design, the selector was hard coded to be a constant 0, such that *a,* the user-inputted *sel* switch, was always outputted.
2. Decoder
   This component's function is basic: To output a two-bit representation of the one bit user input (*sel)*. In other words, the output of the decoder would be 10 if the *sel* switch is off, and 01 if the switch is on.
3. Register
   This component is a direct replica of the JK Flip Flop that was designed in lecture. It has an input clock, a clear, and 4 inputs. Every positive-edge clock cycle, the register saves the previous state and outputs the next. In this scenario, the data was not manipulated as the

component was used to store the user's input data, and pass the relevant data onto the Kelvin and Fahrenheit components. If I was to expand this project in the future, the data stored in this register would be the base for future data manipulation as it would require the user to only input data once.

4. Ripple Carry Adder
   This component relies on four individual Full Adders to add a constant decimal integer of 5, represented as 0101 in Binary to the four input bits. It then returns the addition to represent the Kelvin output. This component varies from the standard model as the final carry out is set to an arbitrary variable because there will never be a carryout issue (project specifications have the maximum input value to be 1010 binary) so it has no meaningful contribution to this component.
5. Truth-Table
   This component's function is to convert a 4-bit input by multiplying it by 7/6 and adding 1 to the result. It was designed by first creating a truth table that matched the above specifications, and the using a K-map to minimize the Boolean function into a SOP function. The black box for this component had 4 inputs, shown in VHDL as a vector of *a* and 4 outputs also shown as a vector *y*. A copy of the truth table and the K-Maps used for each of the outputs can be found in the appendix.

All component mentioned above where then implemented into the main file: *midterm_proj*. The logic in this main circuit was completed by computing each of the conversions each clock cycle such that all data was available. The logic then checks each of the possible conditions of *sel* and *celc*, and outputs the appropriate data. This is efficient as Celsius to Celsius conversions require no intermediate component, and the Fahrenheit and Kelvin calculation are always readily available as they are computed each clock cycle (Which would hypothetically be important for time efficiency if this project were to be expanded in bit size and computation time).

Experimental Data

Seeing as there were no experiments performed in this project, this data is unavailable. A simulation was run on each of the components during the development stage, and a final simulation was run on the entire project. The overall simulation was reviewed and approved by the instructor- A copy of his signature can be found in the appendix.

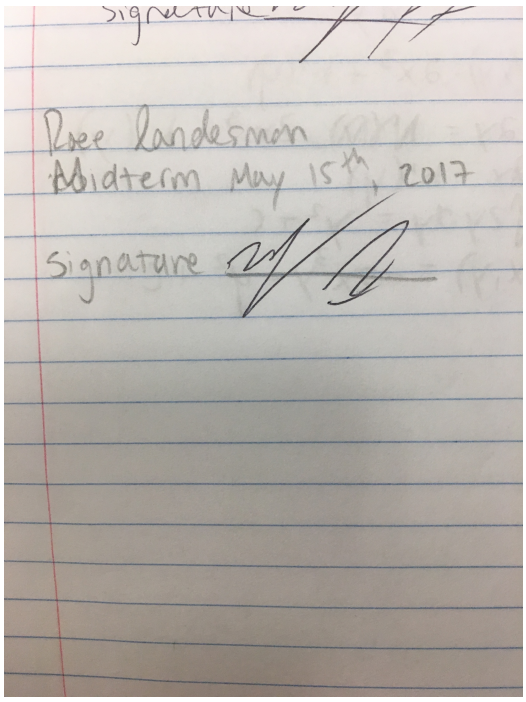
Discussion

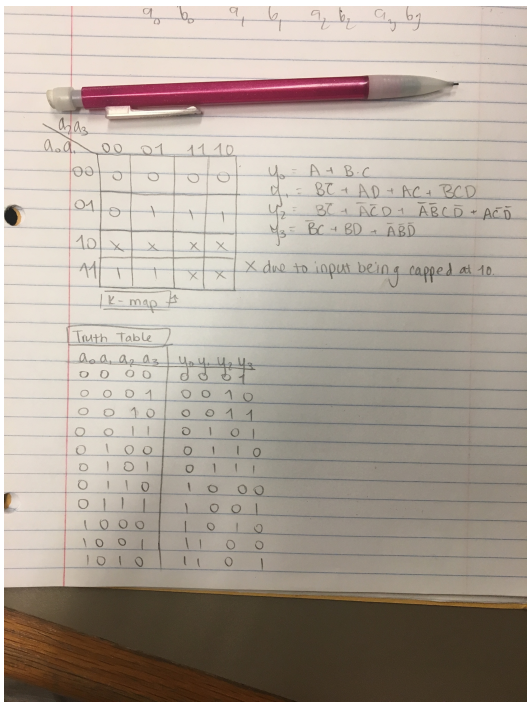
This midterm project had many intriguing facets, and taught me a lot about the technicalities of Digital Design. First and foremost, I learned the importance of simulating and testing every

component before advancing to the next one. Although the complexity of the project was relatively simple, a single faulty component would create issues in the rest of the VHDL code. Furthermore, this project was useful in teaching me the full process from schematic to code, as combining components required a more abstracted understanding of the entire circuit than designing a single component. The board signature is attached below (as a physical copy as I was given an extension by the professor)

Appendix

-Simulation Signature and Hand-Written Fahrenheit Conversion





- Celsius to Fahrenheit - Truth Table Sourced Component

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


entity celc_far is
    Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
           y : out STD_LOGIC_VECTOR (3 downto 0));
end celc_far;

architecture Behavioral of celc_far is

begin

y(0) <= a(0) or (a(1) and a(2));
y(1) <= (a(1) and not a(2)) or (a(0) and a(3)) or (a(0) and a(2)) or (not a(1) and a(2) and a(3));
y(2) <= (a(1) and not a(2)) or (not a(0) and not a(2) and a(3)) or (a(0) and not a(2) and not a(3)) or (not a(0) and not a(1) and
a(2) and not a(3));
y(3) <= (not a(1) and a(2)) or (a(1) and a(3)) or (not a(0) and not a(1) and not a(3));


end Behavioral;
```

- 4-Bit Register

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



entity fbit_reg is
    Port ( D : in STD_LOGIC_VECTOR (3 downto 0);
           clk : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end fbit_reg;

architecture Behavioral of fbit_reg is

begin
process (D, clk)
begin
if (rising_edge(clk)) then

Q(0) <= D(0);
Q(1) <= D(1);
Q(2) <= D(2);
Q(3) <= D(3);

end if;
end process;

end Behavioral;
```

- Selector/MUX

```
entity MUX_str is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           sel : in STD_LOGIC;
           y : out STD_LOGIC);
end MUX_str;

architecture Behavioral of MUX_str is
component and2
port (I1, I2: in std_logic; O1: out std_logic);
end component;
component or2
port (I1, I2: in std_logic; O1: out std_logic);
end component;
component inv2
port (I1: in std_logic; O1: out std_logic);
end component;

signal s0, s1, s2: std_logic;
begin

not1 : inv2 port map(I1 => sel, O1 => s0);
and1 : and2 port map(I1 => a, I2 => sel, O1 => s1);
and3 : and2 port map(I1 => b, I2 => s0, O1 => s2);
or1 : or2 port map(I1 => s1, I2 => s2, O1 => y);

end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity and2 is
port (I1, I2: in std_logic; O1: out std_logic);
end;
architecture and_str of and2 is
begin
O1 <= I1 and I2;
end and_str;


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity or2 is
port (I1, I2: in std_logic; O1: out std_logic);
end;
architecture or_str of or2 is
begin
O1 <= I1 or I2;
end or_str;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity inv2 is
port (I1: in std_logic; O1: out std_logic);
end;
architecture inv_str of inv2 is
begin
O1 <= not I1;
end inv_str;


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

- Decoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity decode_2x1 is
    Port ( IN0 : in STD_LOGIC;
           D1 : out STD_LOGIC;
           D0 : out STD_LOGIC);
end decode_2x1;
architecture Behavioral of decode_2x1 is




begin
D0 <= not IN0;
D1 <= IN0;


end Behavioral;
```

- Ripple Carry Adder (RCA)

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

entity r_c_a_five is
    Port ( cf : in STD_LOGIC_VECTOR (3 downto 0);
           SUMf : out STD_LOGIC_VECTOR (3 downto 0);
           carry_outf : out STD_LOGIC);
end r_c_a_five;

architecture Behavioral of r_c_a_five is
component full_adr
Port ( A : in STD_LOGIC;
       B : in STD_LOGIC;
           C : in STD_LOGIC;
           SUM : out STD_LOGIC;
           CARRY : out STD_LOGIC);
end component;
signal cr0, cr1, cr2, cr7: STD_LOGIC;

begin
FA0: full_adr port map(A => cf(0), B => '1', C => '0', SUM => SUMf(0), CARRY => cr0);
FA2: full_adr port map(A => cf(1), B => '0', C => cr0, SUM => SUMf(1), CARRY => cr1);
FA3: full_adr port map(A => cf(2), B => '1', C => cr1, SUM => SUMf(2), CARRY => cr2);
FA4: full_adr port map(A => cf(3), B => '0', C => cr2, SUM => SUMf(3), CARRY => cr7);


end Behavioral;
```

- Main file!

```
entity midterm_proj is
    Port ( af : in STD_LOGIC_VECTOR (3 downto 0);
           sel : in STD_LOGIC;
           celc : in STD_LOGIC;
           clk : in STD_LOGIC;
           outf : out STD_LOGIC_VECTOR (3 downto 0));
end midterm_proj;

architecture Behavioral of midterm_proj is
component decode_2x1
port (IN0 : in std_logic;
      D1, D0: out std_logic);
end component;

component MUX_str
port (a, b, sel : in std_logic;
     y : out std_logic);
end component;

component r_c_a_five
port (cf: in std_logic_vector;
      SUMf : out std_logic_vector);
end component;

component fbit_reg
port (D : in std_logic_vector;
      clk : in std_logic;
      Q : out std_logic_vector);
end component;

component celc_far
port (a : in std_logic_vector;
      y : out std_logic_vector);
end component;
```

```
signal s0, s1, s2, c2f0, c2f1, c2f2, c2f3, c2k0, c2k1, c2k2, c2k3, r0, r1, r2, r3 : std_logic;
begin

dec1 : decode_2x1 port map(IN0 => sel, D0=> s1, D1 => s0);
mux1 : MUX_str port map(a => s0, b => s1, sel => '0', y => s2);
reg1 : fbit_reg port map(clk => clk, D(3) => af(0), D(2) => af(1), D(1) => af(2), D(0) => af(3), Q(0) => r0, Q(1) => r1, Q(2) => r2,
Q(3) => r3);
K : r_c_a_five port map(cf(0) => af(3), cf(1) => af(2), cf(2) => af(1), cf(3) => af(0), SUMf(0) => c2k3, SUMf(1) => c2k2, SUMf(2) =>
c2k1, SUMf(3) => c2k0);
F : celc_far port map(a(0) => af(0), a(1) => af(1), a(2) => af(2), a(3) => af(3), y(0) => c2f0, y(1) => c2f1, y(2) => c2f2, y(3) =>
c2f3);

process (clk)
begin
if (celc = '1') then
outf(0) <= af(0);
outf(1) <= af(1);
outf(2) <= af(2);
outf(3) <= af(3);
elsif (s2 = '0') then
outf(0) <= c2k0;
outf(1) <= c2k1;
outf(2) <= c2k2;
outf(3) <= c2k3;
elsif (s2 = '1')then
outf(0) <= c2f3;
outf(1) <= c2f2;
outf(2) <= c2f1;
outf(3) <= c2f0;
else
outf(0) <= af(0);
outf(1) <= af(1);
outf(2) <= af(2);
outf(3) <= af(3);

end if;
end process;
```