Chapter 1: Introduction

1.1. Objectives

Temperature sensors play a pivotal role in diverse sectors, such as environmental monitoring, healthcare, automotive, and manufacturing. The ability to precisely measure and monitor temperature empowers industries to maintain optimal conditions, ensure safety, and enhance efficiency.

Our project centers around creating a functional temperature sensing system. This involves selecting suitable temperature sensors, interfacing them with digital circuits, and employing programming languages like VHDL to process the sensor data. By the end of this project, we aim to understand the synergy between physical phenomena, electronic components, and digital design.

The purpose of the project is designing VHDL code, using LM35 temperature sensor, ADC 0808 to convert analog signal from the LM35 sensor to digital signal, Serial Port Monitor v1.5 to convert binary output to decimal output, 2 7-segment LEDs to display temperature measured from the LM35 sensor

Design a realistic simulation circuit using the EPM240T100C5 Chip Board and basic electronic components .

1.2. Requirement

In our pursuit of shaping the ultimate design resembling a tangible product, we have established a set of imperative prerequisites that will govern our approach:

- The envisioned product is a digital system, seamlessly blending hardware and software.

- The utilization of on-board hardware is mandated for generating and manipulating signals within the system.

- The adaptation of the system's operational principles must harmonize with the intrinsic limitations of the hardware components.

- The vehicle for translating our design into reality is VHDL programming, affording us the power to encapsulate intricate functionalities.

1.3. Block diagram

A diagram of a computer system

Description automatically generated

Figure 1. Block diagram

1.4. Component

1.4.1. Altera max II epm 240

The system will be implemented on Altera max II epm 240.

A close-up of a blue circuit board

Description automatically generated

Figure 2. Altera max II epm 240

1.4.2. ADC 0808

We use ADC 0808 to convert analog signal from the sensor to digital signal.



Figure 3. ADC 0808

A black rectangular object with white text

Description automatically generated

Figure 4. Configure pins ADC0808

In this project, we use pin26 IN0 of the ADC0808 to receive the signal from the temperature sensor.

1.4.3. LM35 Sensor

We use temperature sensor LM35 to measure temperature from environment.A close-up of a transistor

Description automatically generated

Figure 5. LM35 Sensor

1.4.4. LED 7 segments

We use LED 7 segments to display the temperature measurement results.

A black and white digital display

Description automatically generated

Figure 6. 7-Segment Display

Chapter 2. System Design

2.1. Led Scanning

2.1.1. Working of Seven Segment LED Display

A seven segment display consists of seven LED segments arranged like a decimal 8. These LED segments are illuminated to form a pattern that represents a decimal number from 0 to 9.

When electrical energy is supplied to all the segments, then the seven segment LED display shows the decimal number 8.

* When power is given to all the segments and if we disconnect power from the segment ‘g’, then it displays the decimal number 0.
* When the power is given to segments "b" and "c" only, then it displays the number 1.
* When the power is given to the segments "a", "b", "g", "e", "d", then it displays the number 2.
* When the power is given to segments "a", "b", "g", "c", "d", then it displays the number 3.
* When the power is given to segments "b", "c", "f", "g", then it displays the number 4.
* When the power is given to segments "a", "c", "d", "f", "g", then it displays the number 5.
* When the power is given to segments "a", "c", "d", "e", "f", "g", then it displays the number 6.
* When the power is given to segments "a", "b", "c", then it displays the number 7.
* When the power is given to segments "a", "b", "c", "d", "f", "g", then it displays the number 9.

In this way, we can display any decimal number from 0 to 9 by illuminating a set of LED segments

2.1.2. Truth Table of Seven Segment LED Display

In our project, we choose Common Anode, if the anode of all 8 LEDs are connected together and the cathode stands individually.

A screenshot of a computer

Description automatically generated

Figure 7. Truth Table of Seven Segment LED Display

2.1.3. Led scanning algorithm

The technique of LED Scanning presents an enhanced approach to optimizing the display of 7-segment LED units. The representation of a single LED utilizing 7 bars necessitates the utilization of 7 data lines (8 if incorporating decimal points). This prompts the query of how to effectively govern the illumination of 2, 3, or 4 LEDs. The intuitive solution might involve expanding the number of data ports, with each LED's 7 bars linked to 7 separate data lines. However, this method raises legitimate reservations during the circuit design phase, prompting numerous inquiries.

The challenge becomes evident when considering that controlling just one 7-segment LED bar mandates a minimum of 8 ports. Practical scenarios encompassing arrays of 7-segment LEDs—like perpetual calendars or gold price listings—exacerbate the issue. Assuming a requirement to display 'n' digits within each system, adhering to the aforementioned technique would necessitate a considerable allocation of resources: 8 times 'n' ports, equivalent to 8 times 'n' data lines for effective control. This would inevitably lead to a system that is unwieldy both in terms of software and hardware, posing significant challenges.

The resolution, in fact, resides in the implementation of the LED Scan method for 7-segment LED displays. The core concept behind this method can be succinctly summarized: "All 7-segment LEDs share a common data line, with only one LED illuminated at any given instance.

[2.2.](https://www.researchgate.net/figure/Actual-Circuit-Diagram_fig4_356728640) Actual Circuit

A circuit board with wires and a digital timer

Description automatically generated

Figure 8. Circuit of the system

CHAPTER 4: PROGRAMMING

The VHDL Code:

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use ieee.std\_logic\_unsigned.all;

use ieee.numeric\_std.all;

entity ngoc is

port (

clk : in std\_logic;

eoc : in std\_logic;

datain : in std\_logic\_vector(7 downto 0);

adc\_clk : out std\_logic;

start\_adc : out std\_logic;

tx : out std\_logic;

led7seg : out std\_logic\_vector(6 downto 0);

D: out std\_logic\_vector(2 downto 1)

);

end entity ngoc;

architecture behavioral of ngoc is

signal bcd: std\_logic\_vector(3 downto 0) := "0101";

---------------signal for adc\_control-----------------------

signal count\_adc: std\_logic\_vector(24 downto 0) := "0000000000000000000000000";

---------------signal for catching rising\_edge of eoc ----------

signal reg: std\_logic\_vector(1 downto 0):= "00";

signal reg\_xor: std\_logic := '0';

signal en : std\_logic;

----------------signal for uart\_transsmit ----------------------

signal data: std\_logic\_vector(7 downto 0) := "01001101";

type state is (idle,start\_bit,data\_bit,stop\_bit);

signal pre\_state: state := idle;

signal count : integer range 0 to 5300;

signal index : integer range 0 to 8;

-------------signal for 7seg-------------

signal int\_val: integer range 0 to 200;

signal a,b: integer range 0 to 9;

signal a\_binary,b\_binary: std\_logic\_vector(3 downto 0);

type step is (waiting, cal0, caldone);

signal step\_n: step := waiting;

begin

---------------------adc control----------------------

start\_adc <= '1' when count\_adc < 6 else '0';

adc\_clk <= count\_adc(11);

adc\_control: process (clk)

begin

if rising\_edge(clk) then

count\_adc <= count\_adc + 1;

end if;

end process adc\_control;

---------------------//-------------------//-----------

-----------------catching rising\_edge of eoc----------

en <= reg\_xor and eoc;

reg\_xor <= reg(1) xor reg (0);

ngoc\_di\_len: process(clk,eoc,reg\_xor)

begin

if rising\_edge(clk) then

reg <= reg(0) & eoc;

end if;

end process ngoc\_di\_len;

------------------//-------------------//-----------------

--------------------uart\_transsmit------------------------

uart\_transsmit: process(clk,en)

begin

if rising\_edge(clk) then

case pre\_state is

when idle =>

tx <= '1';

if en = '1' then

count <= 0;

pre\_state <= start\_bit;

data <= datain;

end if;

when start\_bit =>

tx <= '0';

if count = 5208 then

pre\_state <= data\_bit;

count <= 0;

index <= 0;

else

count <= count + 1;

end if;

when data\_bit =>

tx <= data(index);

count <= count + 1;

if index > 7 then

pre\_state <= stop\_bit;

count <= 0;

index <= 0;

elsif count = 5208 then

count <= 0;

index <= index + 1;

end if;

when stop\_bit =>

tx <= '1';

count <= count + 1;

if count = 5208 then

count <= 0;

pre\_state <= idle;

end if;

end case;

end if;

end process uart\_transsmit;

--------------------bcd covert------------------

process(bcd)

begin

case bcd is

when "0000" => led7seg <= "0000001";

when "0001" => led7seg <= "1001111";

when "0010" => led7seg <= "0010010";

when "0011" => led7seg <= "0000110";

when "0100" => led7seg <= "1001100";

when "0101" => led7seg <= "0100100";

when "0110" => led7seg <= "0100000";

when "0111" => led7seg <= "0001111";

when "1000" => led7seg <= "0000000";

when "1001" => led7seg <= "0000100";

when others => led7seg <= "1111111";

end case;

end process;

----------------------------//-------------------

D <= "01" when count\_adc(12) = '0' else "10";

bcd <= a\_binary when count\_adc(12) = '0' else b\_binary;

process(clk,en)

begin

if rising\_edge(clk) then

case step\_n is

when waiting =>

if en = '1' then

int\_val <= to\_integer(unsigned(datain));

step\_n <= cal0;

end if;

when cal0 =>

a <= int\_val /10;

b <= int\_val mod 10;

step\_n <= caldone;

when caldone =>

a\_binary <= std\_logic\_vector(to\_unsigned(a, 4));

b\_binary <= std\_logic\_vector(to\_unsigned(b, 4));

step\_n <= waiting;

end case;

end if;

end process;

end behavioral;

4.1. Code VHDL for ADC 0808

ADC0808: This VHDL code orchestrates the various control signals and timing required to control the ADC0808 module's operation, ensuring accurate analog-to-digital conversion and data retrieval.

A white background with colorful text

Description automatically generated

Figure 9. Signal for ADC 0808

A screenshot of a computer code

Description automatically generated

Figure 10. Code VHDL for ADC

Block adc\_control:

First we declare a signal count\_adc, this is a 25 bit binary counter. Aim to generate a counter within cycles of the clk pulse. approximately 1s.

The start pulse of adc will be high in tws time (100ns-200ns) so we choose the period as 120ns. we will set start\_adc high when count\_adc counts to 5, that is less than 6, for the rest of the cycle start\_adc is low.

We choose the frequency for adc to be 50khz, because still we will give adc\_clk equal to the state of the 11th bit of count\_adc. adc\_clk will be high for clk cycles, and low for clk cycles.

In the process, each time the clk pulse goes up, the count\_adc variable is added by 1.

Block catching rising\_edge of eoc:

We created a signal reg which is a 2-bit register, initially with the value 00.

signal reg\_xor to compare the value of 2 bits in the signal en register to catch the rising edge of eoc. When the adc receives the start signal, starts the conversion process, the eoc will be pulled low and when the conversion is complete, it will be brought back to high. So every time the clk pulse goes up, we will watch value of eoc, this value will be stored in the low bit of the reg register, the old low value will be pushed to 1 bit. Now reg\_xor will carry the value of xor 2, the value of 2 bits in the register, if there is a difference, then reg\_xor will be 1. The value of en will be the and operation of reg\_xor and eoc, so that en=1 means the end At the end of the transition, reg\_xor is equal to 1 (ie, there is a change in the pulse eoc) and eoc now has the value 1.

4.2. Code VHDL for UART transmirt

UART: The code defines a UART transmitter module that takes an 8-bit data input, initiates transmission when the “start” signal is asserted, and transmits the data with start and stop bits according to the UART protocol. The module operates based on a state machine that controls the timing of signal transitions

A diagram of a diagram

Description automatically generated

Figure 11. Flow chart for UART transmirt

A computer code with numbers and letters

Description automatically generated

Figure 12. . Signal for UART transmirt

A screenshot of a computer code

Description automatically generatedA screenshot of a computer code

Description automatically generated

Figure 13. . Code VHDL for UART transmirt

4.3.Code VHDL for Seven Segment LED

*Led scanning block* : This VHDL code appears to be for two 7-segment LED display driver that sequentially displays data for different decimal places (units, tens, hundreds, thousands). The display is controlled by an external clock signal (clk) and an enable signal (oe). The values for different decimal places are provided as input, and the module multiplexes the display to show each decimal place for a specific period of time.

A computer code with colorful text

Description automatically generated

Figure 14. Signal for 7 Segment LED

A screenshot of a computer code

Description automatically generated

Figure 15. Code VHDL for bcd covert

A screenshot of a computer code

Description automatically generated

Figure 16. : Code VHDL for LEDs scanning

Block 7seg:

signal bcd to represent a 4-bit binary value

int\_val has an integer value from 0-200 because the temperature sensor measures up to 150 degrees

signal a,b are integers from 0-9

a\_binary and b\_binary to convert the value of a,b to a 4-bit binary value.

Declare a new data type named step consisting of 3 values.

Next is a variable step\_n to carry the data type of step, which is initially given the value waiting.

The two numbers on leg7 these 2 numbers do not represent two different values at the same time, so I will show the numbers on each light in turn. D=01 means the left light will be on, for the time the 12th bit of count\_adc is 0, equivalent to about 50ms, for the rest of the time D1 will be off and D2 is on. Similarly, the a\_binary value will also be passed to bcd within the time the 12th bit is zero, the rest of the time the b\_binary value will be passed to bcd.

In this process, each time clk goes up, the state of step\_n will be equal to waiting, if en=1, the conversion is done, the value of the received 8bit datain will be converted to integer value and stored in int\_val.

That will switch to the cal0 state.

The tens digit value will be stored in variable a, the unit row value will be stored in variable b.

Then switch to the caldone state:

The 4-bit binary value of a is stored in a\_binary and the 4-bit binary value of b is stored in b\_binary.

Then it will return to the waiting state to continue waking up.

CONCLUSION

Revolved around FPGA technology and the VHDL programming language, this subject immerses itself in pragmatic applications, placing a strong emphasis on tangible real-world functionality. With the trajectory of technological progress pointing upward, there exists an optimistic outlook for the widespread integration of FPGA and VHDL-based products throughout Vietnam. Over the course of six weeks, the team engaged in extensive collaboration, involving exhaustive research, intricate programming, and rigorous testing protocols. Despite confronting unanticipated hurdles in the initial phases, the team's steadfast commitment to perpetual learning and fearless experimentation paved the way for remarkable advancements. Ultimately, the product was executed with resounding success. The team wishes to extend a heartfelt gratitude to Phd Nguyen Dai Duong, whose unwavering support played a pivotal role in bolstering their comprehension and nurturing their unwavering resolve to bring their envisioned product to fruition.