2023 Project Report

Name	ID
Ahmed Mohamed Ahmed	7197
Ahmed Hazem	7193
Eyad Salama	7128
Hossam Shaybon	6788
Zeyad Zakaria	6764

Digitally Controlled Frequency Generator Project

Team 33

Table of Contents

1.INTRODUCTION

This report outlines the design, implementation, and testing of a project that aims to generate a square wave with different frequencies up to 9999 Hz using AT89S51 microcontroller. The project involves entering the frequency value via a hex keypad and displaying it on a quad seven-segment display. After entering the frequency value, a push button connected to INT0 is pressed to enable the generation of the square wave on pin P3.7.

The report begins by describing the hardware components used in the project and providing a schematic diagram of the circuit. It then goes on to explain the software used in the project, including the Keil compiler and Proteus Simulator, and provides a detailed explanation of the code used.

The design and implementation of the project are then discussed, along with any challenges faced during the process. The results of the project are presented, including the testing process used to verify the functionality of the project.

The report also includes a detailed circuit diagram, programming details, and simulations.

In conclusion, the report summarizes the project and its outcomes, including any challenges faced and lessons learned. It also includes appendices with additional information such as data sheets, and technical specifications.

2.HARDWARE DESCRIPTION

The project utilizes AT89S51 microcontroller to generate a square wave with different frequencies up to 9999 Hz.

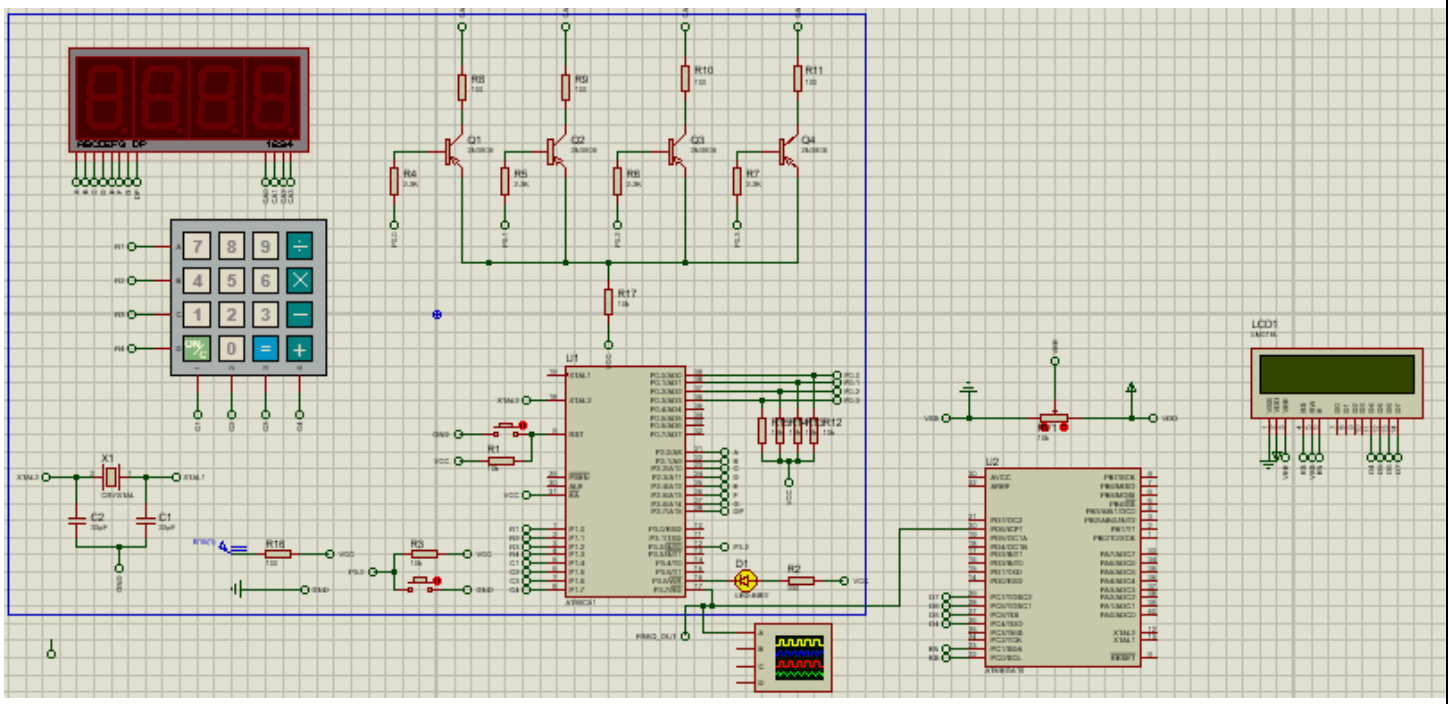
2.1 Components

The following hardware components were used in the project:

1. AT89S51/52 Microcontroller: The microcontroller is the main component of the project and is responsible for controlling the generation of the square wave. It is programmed using the Keil compiler.
2. Hex Keypad: The hex keypad is used to input the frequency value to be generated. It is connected to P1 and is scanned via the software to determine the pressed key from 0 to 9.
3. Quad Seven-Segment Display: The segments of the quad common anode display module are connected to P2, and the control of the four common anodes is done using P0.0 to P0.3 via 2N3906 PNP bipolar transistors. The refreshing of the display is done using Timer 0 interrupt. After pressing a key, it is displayed and shifted until 4 digits are entered.
4. Push Button: A push button connected to INT0 is used to enable the generation of the square wave on pin P3.7. After entering the frequency value, the switch is pressed to generate an external interrupt which loads timer 1 with the proper values and enables square wave generation on pin P3.7.
5. Crystal Oscillator: The crystal used with the project is 11.0592 MHz
6. USB2TTL Module: The USB2TTL module is used for debugging purposes.
7. 33pF Capacitors: Two 33pF capacitors are used to stabilize the crystal oscillator.
8. ATMega16: An ATMega16 is connected to an LCD to display the output frequency.

The circuit diagram of the project is provided in the report, including annotations for each component used. The code used in the project and the simulation results are also included in the report. Additionally, the report discusses the challenges faced during the project and suggests potential future work and improvements to the design and implementation.

2.2 Schematic diagram



3.SOFTWARE DESCRIPTION

The software used in the project is written in C language and is compiled using the Keil compiler. The software is responsible for controlling the input from the hex keypad, refreshing the quad seven-segment display, and generating the square wave output.

3.1 Software Components

The software used in the project is written in C programming language and is compiled using the Keil compiler. The following software components were used in the project:

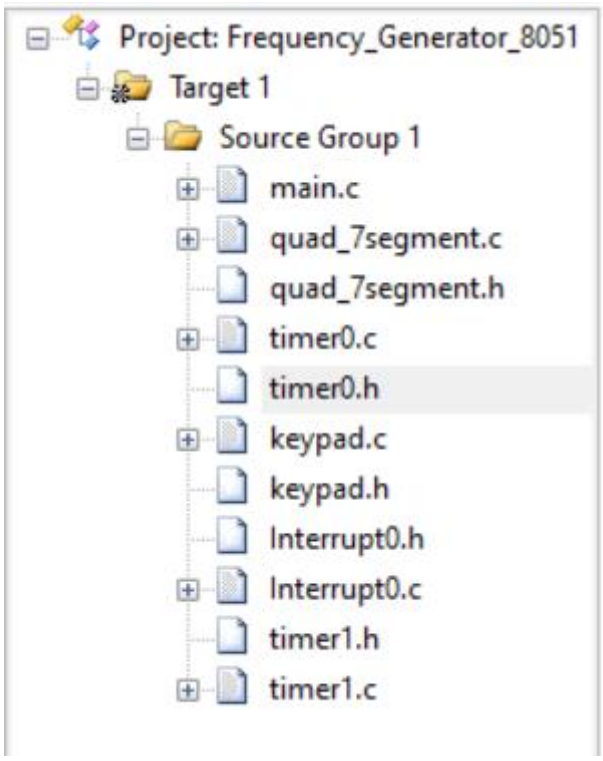
1. Keil Compiler: The Keil compiler is used to compile the C code and generate the hex file that is burned into the microcontroller.
2. Proteus Simulator: The Proteus Simulator is used to simulate the circuit before burning the code into the microcontroller. The simulation helps to identify potential issues and errors before the actual implementation of the circuit.

3. LCD Library for Arduino: An LCD library is used for the Arduino to display the output frequency. The library provides functions that make it easy to interface with the LCD.

4. Timer Interrupt: Timer interrupt is used to refresh the quad seven-segment display. Timer 0 interrupt is used to refresh the display while Timer 1 interrupt is used to generate the square wave.
5. External Interrupt: External interrupt is used to enable the generation of the square wave on pin P3.7. When the push button connected to INT0 is pressed, an external interrupt is generated, and Timer 1 is loaded with the proper values to generate the square wave.
6. Keypad Scanning: The hex keypad is scanned via the software to determine the pressed key from 0 to 9. The scanned values are then displayed on the quad seven-segment display.

The code used in the project is provided in the report, including annotations for each function and section of the code. The simulation results are also included in the report, showing the output of the circuit for various input frequencies.

3.2 Code



Quad_7segment.c

This code is a C program for driving a 4-digit 7-segment display using an 8051 microcontroller. The program includes functions for initializing the display, converting numerical digits to their corresponding 7-segment patterns, and displaying a given number on the display.

The program uses timer0 to update the display at a regular interval. The display is connected to the microcontroller using four control pins (P0.0 - P0.3) and a data pin (P2.0 - P2.7).

```
void init_7segment() {
  init_timer0();
  control1 = 0;  // turn off control
  control2 = 0;  // turn off control
  control3 = 0;  // turn off control
  control4 = 0;  // turn off control
  P2 = DIGIT_4;  // clear PORT
}
```

The init_7segment() function initializes the control pins and clears the display by setting the data pin to DIGIT_4, which is a blank pattern.

```
unsigned char numToPattern(unsigned char digit) {
  unsigned char pattern = 0;
  switch (digit) {
    case 0:
      pattern = DIGIT_0;
      break;
    case 1:
      pattern = DIGIT_1;
      break;
    case 2:
      pattern = DIGIT_2;
      break;
    case 3:
      pattern = DIGIT_3;
      break;
    case 4:
      pattern = DIGIT_4;
      break;
    case 5:
      pattern = DIGIT_5;
      break;
    case 6:
      pattern = DIGIT_6;
      break;
    case 7:
      pattern = DIGIT_7;
      break;
    case 8:
      pattern = DIGIT_8;
      break;
    case 9:
      pattern = DIGIT_9;
      break;
    default:
      // Display a blank segment
      pattern = BLANK;
      break;
  }
  return pattern;
}
```

The numToPattern() function takes a numerical digit as input and returns the corresponding 7-segment pattern. If the input digit is not in the range of 0-9, the function returns a blank pattern.

```
void show_7segment(unsigned int num) {
  // 1 2 3 4
  unsigned char digit1 = num % 10;
  unsigned char digit2 = (num / 10) % 10;
  unsigned char digit3 = (num / 100) % 10;
  unsigned char digit4 = (num / 1000) % 10;
  unsigned char pattern1 = numToPattern(digit1);
  unsigned char pattern2 = numToPattern(digit2);
  unsigned char pattern3 = numToPattern(digit3);
  unsigned char pattern4 = numToPattern(digit4);
  int flag = get_global_variable();
  if (flag == 0) {
    control1 = 1;
    control2 = 0;
    control3 = 0;
    control4 = 0;
    P2 = pattern1;
  } else if (flag == 1) {
    control1 = 0;
    control2 = 1;
    control3 = 0;
    control4 = 0;
    P2 = pattern2;
  } else if (flag == 2) {
    control1 = 0;
    control2 = 0;
    control3 = 1;
    control4 = 0;
    P2 = pattern3;
  } else if (flag == 3) {
    control1 = 0;
    control2 = 0;
    control3 = 0;
    control4 = 1;
    P2 = pattern4;
  } else if (flag == 4) {
    control1 = 0;
    control2 = 0;
    control3 = 0;
    control4 = 0;
    P2 = pattern4;
  }
}
```

The show_7segment() function takes an integer number as input and displays it on the 4-digit 7-segment display. It first breaks down the number into four individual digits and converts each digit to its corresponding pattern using the numToPattern() function. It then uses a global variable to cycle through the four digits, displaying each one for a short period of time. The global variable is updated by the timer0 interrupt service routine, which is not shown in this code.

Overall, this code demonstrates a basic implementation of a 4-digit 7-segment display using an 8051 microcontroller and C programming language.

Keypad.c

```
unsigned char read_char(void) {
  unsigned char key = 'k';  // k is a flag
  ROW1 = 0;
  ROW2 = 1;
  ROW3 = 1;
  ROW4 = 1;
  while (COL1 == 0) key = '1';
  while (COL2 == 0) key = '2';
  while (COL3 == 0) key = '3';
  while (COL4 == 0) key = 'C';
  ROW1 = 1;
  ROW2 = 0;
  ROW3 = 1;
  ROW4 = 1;
  while (COL1 == 0) key = '4';
  while (COL2 == 0) key = '5';
  while (COL3 == 0) key = '6';
  while (COL4 == 0) key = 'B';
  ROW1 = 1;
  ROW2 = 1;
  ROW3 = 0;
  ROW4 = 1;
  while (COL1 == 0) key = '7';
  while (COL2 == 0) key = '8';
  while (COL3 == 0) key = '9';
  while (COL4 == 0) key = 'A';
  ROW1 = 1;
  ROW2 = 1;
  ROW3 = 1;
  ROW4 = 0;
  while (COL1 == 0) key = '*';
  while (COL2 == 0) key = '0';
  while (COL3 == 0) key = '#';
  while (COL4 == 0) key = 'D';
  return key;
}
```

This code implements a basic keypad interface using a 4x4 matrix of buttons. It initializes the necessary I/O pins as inputs, and provides a function read_char() that scans the matrix and returns the character corresponding to the button that was pressed.

The rows of the matrix are connected to pins P1.0-P1.3, and the columns are connected to pins P1.4-P1.7. The read_char() function works by setting one row at a time to a low logic level, and then checking the state of each column in turn. If a column is found to be low, the corresponding button was pressed, and the corresponding character value is returned. The function uses a flag value of 'k' to indicate that no button was pressed, in case multiple buttons are pressed simultaneously or there is no button pressed.

Main.c

```
void main(void) {
  // Intialize frequency "num"
  unsigned int num = 0;
  // Intialize counter
  short counter = 0;

  // Intialize 7-Segment
  init_7segment();
  // Intialize keypad
  keypad_init();
  // Intialize Interrupts 0
  Ext_int0_Init();

  while (1) {
    // read character from keypad
    unsigned char temp = read_char();
    // if it is k skip otherwise enter if condition
    if (temp != 'k') {
      // multiply num by 10
      num = num * 10;
      // Add newly read character to the 7 segment
      num += (temp - '0');
      // Counter to check if limit reached
      counter++;
    }
    // print on 7 segment
    // Global Flag
    //  Frequency generator
    show_7segment(num);  // 1 2 3 4
    if (flag == 0) {
      set_sq_wave(num);
    }
  }
}
```

This code initializes and runs a program on an 8051 microcontroller to read input from a keypad, display it on a 7-segment display, and generate a square wave of frequency based on the input. The main function initializes and sets up the program by first initializing the necessary libraries and variables. It initializes the 7-segment display, keypad, and Interrupt 0.

Inside the infinite while loop, it reads input from the keypad, multiplies the previous value of the number by 10 and adds the newly read character to the number. The counter variable keeps track of how many characters have been entered. It then displays the number on the 7-segment display and checks if the global flag is set to 0. If the flag is 0, it generates a square wave of frequency based on the number entered by calling the set_sq_wave() function from the timer0 library. This loop continues indefinitely until the program is reset or powered off.

Timer0.c

This code implements Timer 0 with interrupt in 16-bit mode. The function init_timer0() initializes Timer 0 by setting the timer mode to 16-bit mode and enabling the timer interrupt. It also enables interrupts globally and starts Timer 0.

The interrupt service routine for Timer 0 is declared as Timer0_INT(). This ISR is executed every time Timer 0 overflows. Inside the ISR, the interrupt flag is reset, and Timer 0 is stopped. Then, the reload value for Timer 0 is set, and a global variable global_variable is assigned a value based on the value of temp. The value of temp is then incremented and checked if it has reached 5, in which case it is reset to 0. Finally, Timer 0 is started again.

This code does not have any practical use, as it only updates a global variable based on a counter. It might be used for testing purposes or as a template for implementing more complex Timer 0 functionality.

Timer1.c

This code implements a square wave generator using Timer 1 on an 8051 microcontroller. The frequency of the square wave can be set using the set_sq_wave function, which takes an integer argument representing the desired frequency in Hertz.

The init_timer1 function initializes Timer 1 to operate in 16-bit mode and enables its interrupt. The start_timer1 function sets the timer reload value to the values stored in the TH and TL variables, and starts the timer by setting the TR1 bit to 1. The stop_timer1 function stops the timer by setting the TR1 bit to 0 and resetting the timer interrupt flag.

The delay_timer1 function is the interrupt service routine for Timer 1, and is called whenever the timer overflows. This function toggles a bit to produce the square wave, and then stops and restarts Timer 1 with the reload value stored in TH and TL.

The set_sq_wave function first checks if the desired frequency is different from the current frequency. If it is, the function calculates the number of timer ticks needed for each high/low portion of the square wave, taking into account the maximum value of 65536 for the timer register. It also calculates an external loop count to ensure that the total loop count does not exceed 65536. The function then sets the timer reload value and starts Timer 1 by calling start_timer1. If the desired frequency is 0, the function stops the square wave generation by calling stop_timer1 and setting the mybit variable to 0.

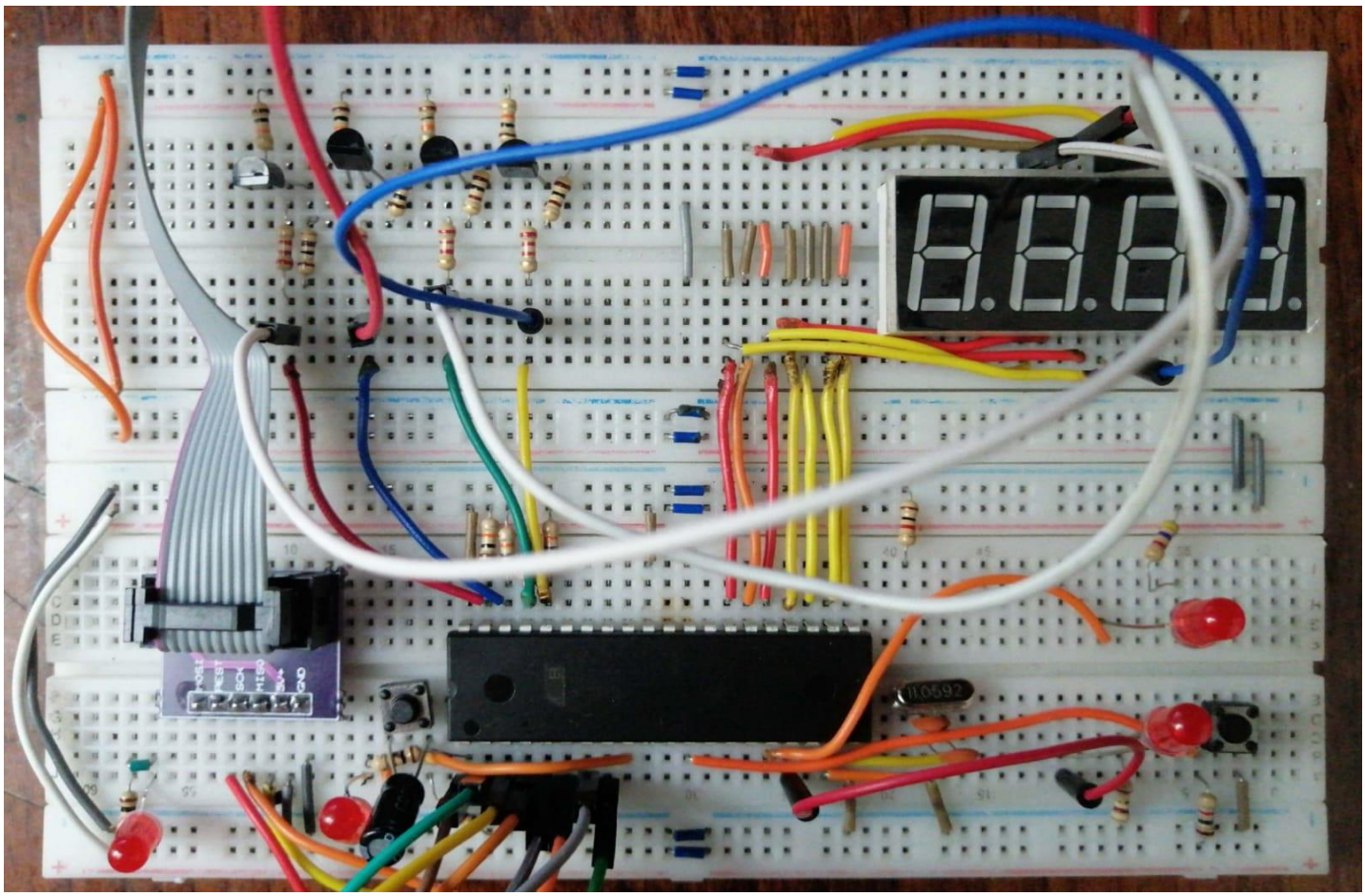
Interrupt0.c

This code initializes and sets up an external interrupt on the microcontroller's interrupt 0 pin (INT0), which is connected to a physical button or switch. When the button is pressed, a falling edge is detected on INT0, which triggers the interrupt and executes the interrupt service routine (ISR) External0_ISR().

The External0_ISR() simply toggles the state of a flag variable by performing a bitwise NOT operation on it. This can be used as a way to signal to the main program that the button has been pressed and to take some action accordingly.

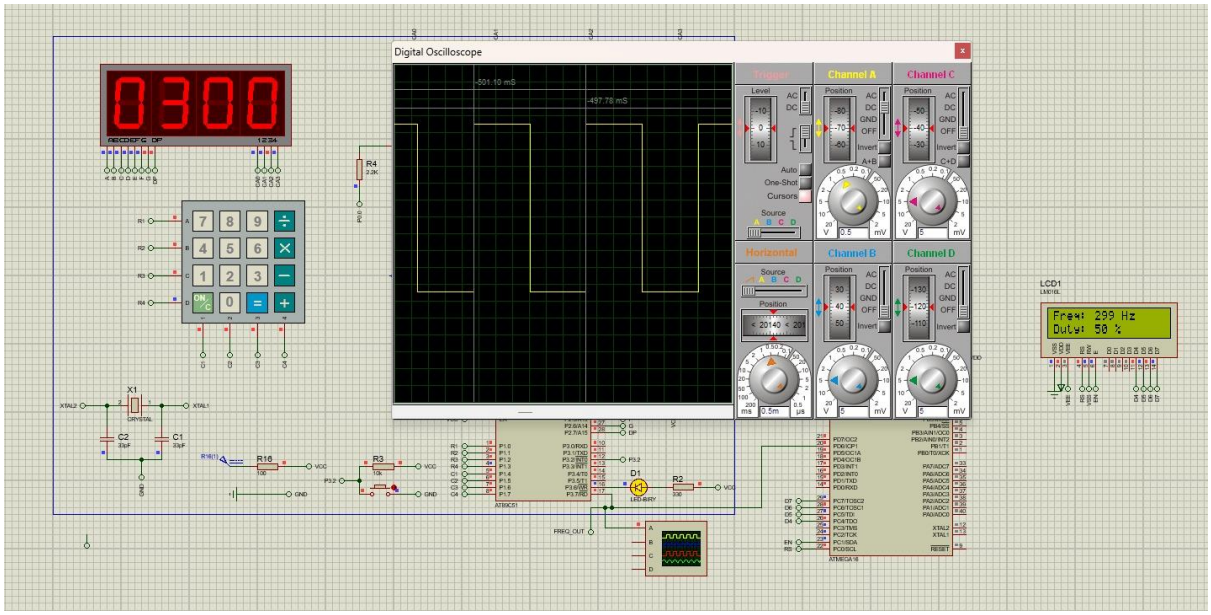
The Ext_int0_Init() function initializes the interrupt by setting up the global interrupt enable bit (EA), enabling interrupt 0 (EX0), and selecting the interrupt trigger type (falling edge in this case) with the IT0 bit. The function also sets the initial value of the flag variable to 1.

4.CIRCUIT DIAGRAM

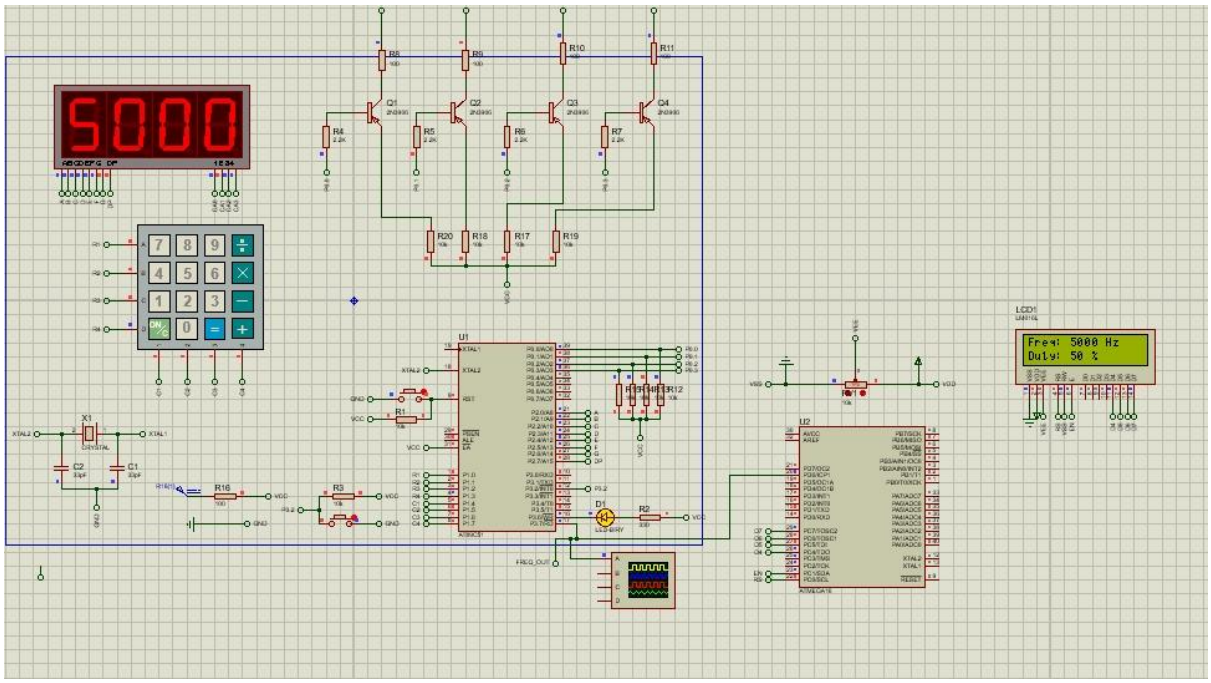


5.RESULTS AND TESTING

Test 300HZ:



Test 5000 HZ:



6. CONCLUSION

In conclusion, the microcontroller-based project successfully achieved its objective of generating a square wave with different frequencies up to 9999 Hz using a hex keypad, quad seven segment display module, and push button connected to INT0. The hardware setup and software implementation were carefully designed to ensure the project's smooth operation, and the use of the Proteus simulator allowed for efficient debugging and troubleshooting. The project also highlighted the importance of careful timing and interrupt management, and provided opportunities to gain valuable insights into microcontroller-based circuit design. Overall, the project demonstrated a successful implementation of a complex system using an AT89S51 microcontroller and provided a foundation for future work and improvements.

7.APPENDICIES

Code listing:

https://drive.google.com/file/d/1GNF21uRsxssSUSN7zr9NS5QnfXGt0Xyj/view?usp=share_link

Proteus Diagram:

https://drive.google.com/file/d/12XZmXhnx2j3UVaD0OLLK0SFtIdDKZBg6/view?usp=share_link

Proteus Simulation Results:

https://drive.google.com/file/d/1_EdJjeeLr656ERF8f42ta3poA-idE9Rc/view?usp=share_link

Test Results:

https://drive.google.com/file/d/1ueRBCCunF3vwRAQl1g_jv8GK-_M0P7Vm/view?usp=share_link

Circuit diagram:

https://drive.google.com/file/d/1Xv4tUNfgz4J0y_a75kXIPfuZr1LuPNs_/view?usp=share_link

8.CONTACT

If you have any questions or comments about this project, please feel free to contact us:

Ahmed Salem

Email: a.salem3214@gmail.com

Ahmed Hazem

Email: ahmadrobotic22@gmail.com

Eyad Salama

Email: eyad41mohamed542@gmail.com

Hossam Shaybon

Email: hossamashraf710@gmail.com

Zeyad Zakria

Email: zeyad.zakaria01@gmail.com

We appreciate your interest in our project and look forward to hearing from you.