Frequency Generator

BY:

Nourhan Mohamed Ismail	7153
Salma Mohamed Hasan Barakat	6845
Rowan Samy Gamal El-din Aly	6733
Mohammad Tarek Helaly	6870

This is a C program for controlling a quad 7-segment with a common anode display and a keypad using an 8051 microcontroller.

1- Variables definitions:

```
#include <reg51.h>
#define KEYPAD_NO_NEW_DATA '-'
static unsigned char Last_valid_Key_G = KEYPAD_NO_NEW_DATA;
unsigned char pKey; // stored inside it the pressed key.
unsigned char position=0; // shows which digits will appear in the 7 segement.
unsigned char IN[]= {'0','0','0','0'}; // array of digits in entered by user.
unsigned char arr[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x98}; // array contains the values to display numbers 0-9 on the 7-segment LED displa

unsigned char highVal= 0x00; // stores the upper 2 hex char of the calculated freq.
unsigned char lowVal = 0x00; // stores the lower 2 hex char of the calculated freq.
unsigned char counter=0; // needed for timers with time more than 71ms.
unsigned char temp=0; // needed for timers with time more than 71ms. decremented till zero and refilled again with counter.

// bin assignment.
sbit LED = P3^7;
sbit Tr1 = P0^0;
sbit Tr2 = P0^1;
sbit Tr3 = P0^2;
sbit Tr4 = P0^3;
sbit c3 =P1^7;
sbit c2 =P1^6;
sbit c1 =P1^5;
sbit c0 =P1^4;
sbit r3 =P1^3;
sbit r2 =P1^2;
sbit r1 =P1^1;
sbit r0 =P1^0;

```

- The program starts with one header file which is reg51. The program then declares some variables and initializes them.
- The 'sbit' statements define the input/output ports used to interface the quad 7-segment common anode display and the keypad with the microcontroller.
- The 'arr' array contains the values to display numbers 0-9 on the quad 7-segment common anode display.
- Some variables are defined to read the keypad inputs including 'pKey' that stores the value of the pressed key.

2- Function “debounce()”:

```

// debounce function used to get some delay after keypad button click.
void debounce()
{
      unsigned char i = 100;
      unsigned char j = 1000;
      for(;i>0;i--)
      for(;j>0;j--);
}

```

- Debouncing is a technique used to remove false signals from a signal line, caused by mechanical switches or other sources of electrical noise. It is commonly used in digital systems to ensure accurate signal readings.

- This debounce function uses two nested for-loops to create a delay of approximately 100 microseconds. This delay helps to stabilize the input signal, by allowing any electrical noise to settle down before the input signal is read.
- The function is called after a keypad input is detected, to ensure that the input signal is stable before the value is read and processed. The debounce function is commonly used in keypad and button input applications to ensure that only valid key presses are detected.

3- Functoin "KeyPad_Scan()":

```
// KeyPad_Scan used to detect the pressed value.
char KeyPad_Scan(void)
{
  static char Old_Key;
  char Key = KEYPAD_NO_NEW_DATA; //-

  //if row zerois pressed check which column is pressed to get the exact letter.
  r0=0;
  if(c3 ==0) {Key= 'A';}
  else if(c2==0) {Key= '3';}
  else if(c1==0) {Key= '2';}
  else if(c0==0) {Key= '1';}
  r0=1;

  r1=0;
  if(c3 ==0) {Key= 'B';}
  else if(c2==0) {Key= '6';}
  else if(c1==0) {Key= '5';}
  else if(c0==0) {Key= '4';}
  r1=1;

  r2=0;
  if(c3 ==0) {Key= 'C';}
  else if(c2==0) {Key= '9';}
  else if(c1==0) {Key= '8';}
  else if(c0==0) {Key= '7';}
  r2=1;
```

```
  r3=0;
  if(c3 ==0) {Key= 'D';}
  else if(c2==0) {Key= '#';}
  else if(c1==0) {Key= '0';}
  else if(c0==0) {Key= '*';}
  r3=1;

  // used to get delay after each press
  debounce();

  // check the validity of the pressed key.
  if(Key == KEYPAD_NO_NEW_DATA)
  {
    Old_Key = KEYPAD_NO_NEW_DATA;
    Last_valid_Key_G = KEYPAD_NO_NEW_DATA;
    return 0;
  }

  // check that the key is not as the last pressed key and assign the global variable pkey.
  if(Key == Old_Key)
  {
    if(Key !=Last_valid_Key_G)
    {
      pKey=Key;
      Last_valid_Key_G = Key;
      return 1;
    }
  }
  Old_Key = Key;
  return 0;
}
```

- This function is used for scanning a 4x4 hex keypad.

- The function reads the keypad matrix by scanning each row (r0 to r3) and checking which column (c0 to c3) has a key pressed. If a key is pressed, it assigns a corresponding character to the variable 'Key'. If no key is pressed, the function returns the value 'KEYPAD_NO_NEW_DATA'.

- The function also uses debounce function to ensure stable readings from the keypad.

- The function checks if the key value has changed from the previous scan which is stored in the variable 'Old_Key'. If the key value has not changed, the function checks if the key has already been read which is stored in the variable 'Last_valid_Key_G'. If the key has not been read, the function stores the key value in 'pKey' and returns 1 to indicate that a new key has been detected. If the key has already been read or if no key is detected, the function returns 0.

4- Function "Timer1_freqGenration()":

```
98  // delay to produce square wave with given time.
99  void Timer1_freqGenration (void) interrupt 3
100 {
101   TH1=highVal;
102   TL1=lowVal;
103   if (temp==0)
104   {LED = ~LED;
105     temp=counter;
106   }
107   else  temp--;
108 }
```

- This is an interrupt service routine for the Timer1 interrupt.

- The Timer1 interrupt is triggered when the Timer1 register overflows. If the delayTime is greater than 71ms then the Timer1 register is set to overflow every 10ms, which means that this interrupt routine will be executed approximately every 10ms if delayTime remained greater than 71ms.

- Inside the interrupt routine, the first action is to reload the Timer1 register with the values of 'highVal' and 'lowVal'. This is done to reset the timer and prepare it for the next overflow interrupt.

- The code also toggles the state of an LED connected to pin P3.7 of the microcontroller. This is done by using the bitwise NOT operator to invert the current state of the LED.

- Finally, we use the variables 'temp' and 'counter' to implement a delay. When the interrupt routine is first executed, 'temp' is set to 0, and the LED is toggled. 'temp' is then set to the value of 'counter'. On subsequent executions of the interrupt routine, 'temp' is decremented. When 'temp' reaches 0, the LED is toggled again, and 'temp' is set back to 'counter'. This creates a delay of 'counter' * 10ms between toggles of the LED.

5- Function “freqGenerator()”:

```
// function to calculate the frequency generator.
void freqGenerator(void ){
  unsigned short int delay;
  float delayTime;
  unsigned short int freq;
  //float adjust;
  freq=(IN[0]-'0')*1000+(IN[1]-'0')*100+(IN[2]-'0')*10+(IN[3]-'0'); // get the frequency the user entered
  // calculate the overhead
  if (freq < 1500)
    delayTime = 0.99;
    else if (freq < 2500)
    delayTime = 0.96;
    else if (freq<3500)
    delayTime = 0.94;
    else if (freq<4500)
    delayTime = 0.92;
    else if (freq<5500)
    delayTime = 0.90;
    else if (freq<7500)
    delayTime = 0.86;
    else if (freq<9500)
    delayTime = 0.83;
    else
    delayTime = 0.80;
```

```
   delayTime = (delayTime/(freq*2));  // delay = Ts/2
  if (delayTime <= 0.071){ //maximum limit at TH0 = 00 and TL0 = 00
    delayTime = delayTime/(0.000001085);
    delayTime = 65536 - delayTime; // get the numbers of cycles.
    delay = (int)delayTime;
    lowVal = delay & 0xff;  // lower 8 bits
    highVal = delay >> 8;   // higher 8 bits
    counter=0;
    temp=0;
  }
  else
    {
      // if the time needed more than 71ms divide the time needed by 10ms and assign it to counter and temp
      counter = (int)(delayTime / 0.01);
      // assign the lowVal and highVal to 10ms if we need 50ms timer then counter equals 5 and loop over 10ms 5 times
      lowVal = 0x0FF;
      highVal = 0x0DB;
      temp=counter;
  }
}
```

- The function freqGenerator() calculates a delay for the given frequency. The function takes input from the IN array, which represents the frequency to be generated in Hertz.

- To compensate for the overhead, the frequency is first adjusted using a conditional statement based on its value to ensure the generated square wave has the required frequency.
- The delay time for the timer is then calculated using the adjusted frequency, and a check is made to ensure the delay time is within the range that the timer can handle.
- If the delay time is within range, the timer values are set to generate a square wave of the required frequency. If not, the timer is set to generate a square wave of 10ms delay and repeats the generation of this delay according in the counter.

6- Function "keyPressed()":

```
156  // called when the user presses the push button.
157  void keyPressed (void) interrupt 0
158  {
159    EA=0;
160    // Calculate the frequency entered by the user.
161    freqGenerator();
162    // Start the timer
163    TR1=1;
164    EA=1;
165  }
```

- This function defines an interrupt service routine for the external interrupt 0 (INT0), which is triggered when the push button is pressed.

- Within keyPressed(), the first action is to disable any other interrupts (EA=0). This is done to prevent any other interrupts from being triggered while the current one is being processed.

- Then, the freqGenerator() function is called, which calculates the delay needed to generate the desired frequency and sets the values of the timer registers (TH1 and TL1) accordingly. The TR1 bit is also set to start the timer.

- Finally, any other interrupts are enabled again (EA=1) before the keyPressed() function returns.

7- Function “display()”:

```
167  // function for timer 0 to display on the 7 segment.
168  void display() interrupt 1
169  {
170    // digits array used to declare Tr1,Tr2,Tr3 and Tr4 with 1 to be closed.
171     unsigned char digits[] = {1,1,1,1};
172    // timer for 4.4ms refreshing screen time.
173      TL0=0x00;
174      TH0=0xF0;
175    // open only 1 Tr to display a digit.
176      digits[position] = 0;
177      Tr1=digits[0];
178      Tr2=digits[1];
179      Tr3=digits[2];
180      Tr4=digits[3];
181    // display on port2.
182      P2 = arr[IN[position]-'0'];
183      position++;
184      if (position == 4){
185          position = 0;
186      }
187  }
```

- This is an interrupt service routine for displaying the input frequency on a 7-segment display. It uses an interrupt to update the display periodically. Here is a breakdown of the code:
- unsigned char position=0; This initializes a variable position to 0. This variable is used to keep track of the current digit being displayed on the 7-segment display.
- TL0=0x00; TH0=0xF0; This sets the reload value for Timer0. The timer will count down from 0xF000 to 0x0000 and then generate an interrupt. This is done to generate a periodic interrupt with a period of 4.4ms.
- P0 represents the digits [] array, which controls the common anode pins of the quad 7-segment display. Each index of the digits [] array is set to 1 initially, and then set to 0 for the corresponding digit to be displayed.
- P2 = arr [IN[position]-'0'];: This sets the appropriate segments on the quad 7-segment display to display the current digit.
- position++; This increments position to select the next digit on the 7-segment display.
  if (position == 4) {position = 0;}: This checks if all digits have been displayed and resets position to 0 to start over.

8- Function “main()”:

```
189  void main()
190  {
191    SP=0x30;
192    // Declare the timer rgister.
193    TMOD =  (TMOD & 0xF0) | 0x11;   // timer 0 in mode 1 (16 bit counter) and timer 1 in mode 1.
194    //  declare timer0 registers
195    IT0=1;
196    ET0=1;
197    TL0=0x00;
198    TH0=0x0F0;
199    // declare int0 registers.
200    EX0=1;
201    EA=1;
202    // start timer0
203    TR0=1;
204    TH1=highVal;
205    TL1=lowVal;
206    ET1=1;
207    EA=1;
208    // declare all rows by zero.
209    r0 = 1;
210    r1 = 1;
211    r2 = 1;
212    r3 = 1;
213
```

```
214    while(1)
215      {
216    // call keypad to detect if user pressed any letter
217      if(KeyPad_Scan() == 1)
218      {
219        // shift the 7 segments letters on click.
220        IN[0]=IN[1];
221        IN[1]=IN[2];
222        IN[2]=IN[3];
223        IN[3]=pKey;
224      }
225    }
226  }
```

- The code sets up timer, interrupts, and KeyPad_Scan(). The timer is set up in mode 1, which is a 16-bit counter. The interrupt 0 is enabled with the EX0=1, statement. Interrupt 1 is also enabled with the ET1=1, statement.

- In the while (1) loop, the code scans the keypad for a key press with the KeyPad_Scan() function. If a key is pressed, the code shifts the digits to the left in “IN” array.

9- Proteus connection: