Tone Generation using PIC18F microcontroller

[1]Sumeet Kuvelkar, [2]Omkar Magar, [3]Akshay Mohite, [4]Varsha K. Patil
[1]Pursuing BE, [2] Pursuing BE, [3]Pursuing BE, [4] Professor
Department of Electronics and Telecommunication,
AISSMS Institute of Information Technology, Pune, India.

Abstract: This paper explains a method to generate sounds of various frequencies or tones using the timer module of a PIC18F Series microcontroller. The microcontroller is interfaced with a speaker driven by an audio frequency switching transistor. The microcontroller uses a Timer 0 module to generate signals of varying frequencies. These frequency signals are fed into the transistorized driver circuit which drives the speaker to emit the different tones generated.

***IndexTerms* – PIC18F, Timer, Frequency, Speaker**

I. Introduction

The timing module in any microcontroller is the one of its most important parts. The timing section consisting of hardware timers provides a consistent heartbeat to all the internal operations of the controller. If this module is mastered, almost any application can be mastered on a microcontroller. This system uses the timer module of the microchip pic18F microcontroller to generate an array of different frequencies and as a practical application of the frequency generation, a speaker is driven to emit different tones representing the different frequencies fed to it. Generation of different frequencies is important because many applications require frequency-varied signals to be generated in order for them to work. So this system can be adapted to a vast array of applications directly with very minor changes mostly in the signal driving systems.

***Index Terms* - PIC18F, Timer, Frequencies, Tone Generation, Speaker.**

II. Hardware

A. Aurum PIC18F development board:

Aurum v1.2 is a PIC18F4550-based board developed at e-Prayog, WEL Lab, IIT Bombay under the Virtual Labs Project by the Ministry of Human Resource and Development (MHRD), Government of India. This is a low-cost solution aimed to cater to the needs of an undergraduate electrical/electronics engineering student in a course in microprocessors, microcontrollers and embedded systems. This board is programmable over USB using a boot loader on PC side as well as a boot loader HEX file dumped on the controller. This makes the programming process very easy and quick leading to fast prototyping time.

Following are the features on Aurum as taken from [1]

- A board centered on PIC18F4550, an 8bit microcontroller by Microchip Technology.
- USB powered and programmable. Additionally, a provision to power through external power supply is also provided.
- Large number of on-board I/Os (31) are provided for various applications.
- Connectors are made available for the interfacing of standard I/O such as switches/LED and LCD.
- Compatible with standard Microchip Technology software tools (MPLAB IDE and HID Bootloader).

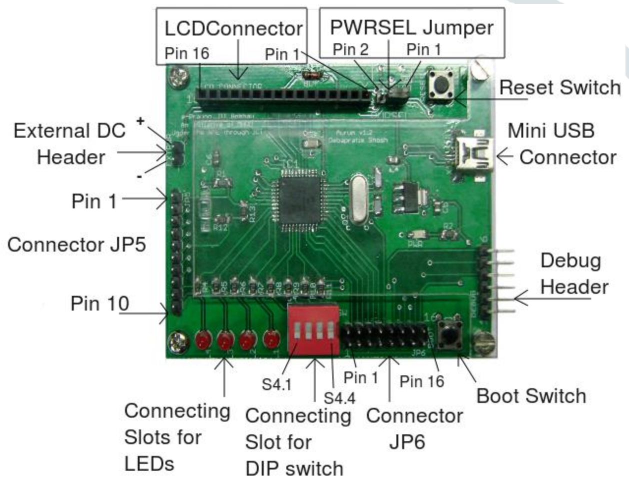


Figure 1: Aurum Board [1]

B. Driver Circuit

The current on the output pin of a microcontroller is not enough to drive a load such as a speaker. Hence we need a transistorized driver circuit to drive the speaker on the output of the microcontroller. This system uses a BC110 transistor circuit due to its high switching frequency and low switching power dissipation [2].

C. Speaker

The speaker used in this project is a 5W 8 ohm speaker. This kind of speaker was chosen because it is easily available and avoids impedance matching problems.

III. Circuit Diagram

As shown in fig.2, the frequency signal is generated on pin RA4 of pic18f4550 microcontroller. Port A is chosen because it does not have internal pull up resistors which improves its driving capability. The output pin is connected to the base of BC110 NPN transistor in series with a 1kohm resistor to generate enough current to drive the base. The Speaker's positive terminal is connected to VCC in series with a 200 ohm resistor to limit current and its negative terminal is connected to the collector of the transistor. In this way, when the base receives a current pulse, the collector is shorted to emitter connected to ground and the speaker's negative is connected to ground making the speaker generate a sound. The tone of this sound is representative of the frequency at which the transistor is switched by the controller.

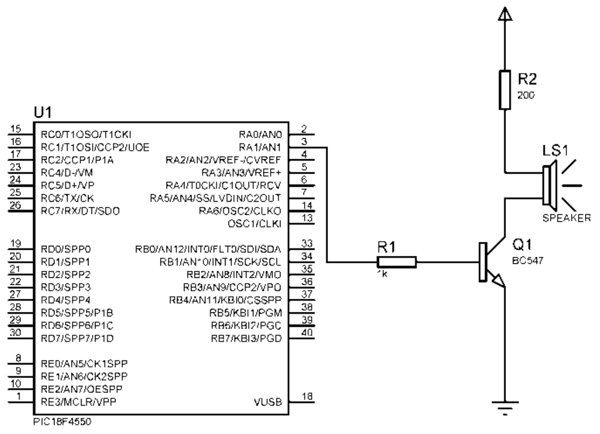


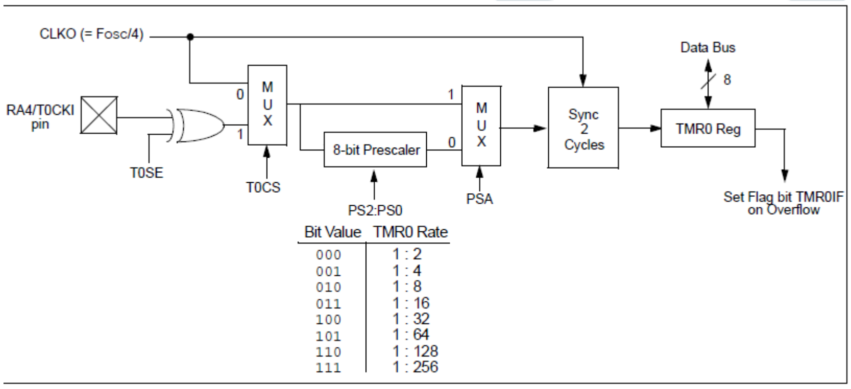
Figure 2: Circuit Diagram

IV. Methodology

A. Timer Operation

The operation of the timer is simple. A timer is a peripheral that counts the internal machine cycles of the controller. The timer has an 8- bit or 16 bit register depending on the configuration. The value of this register increments each machine cycle. When the value reaches its max, the timer overflows and a flag sets. All the user has to do is to poll this flag or enable a timer interrupt to notify the CPU of an overflow.

The prescaler is a way to increase max timer duration while sacrificing resolution. If the prescaler is set to 2, it will divide the machine clock by 2, meaning that the timer register will increment by one every two machine cycles instead of every one machine cycle. Timer operation in shown in figure 3. [3]

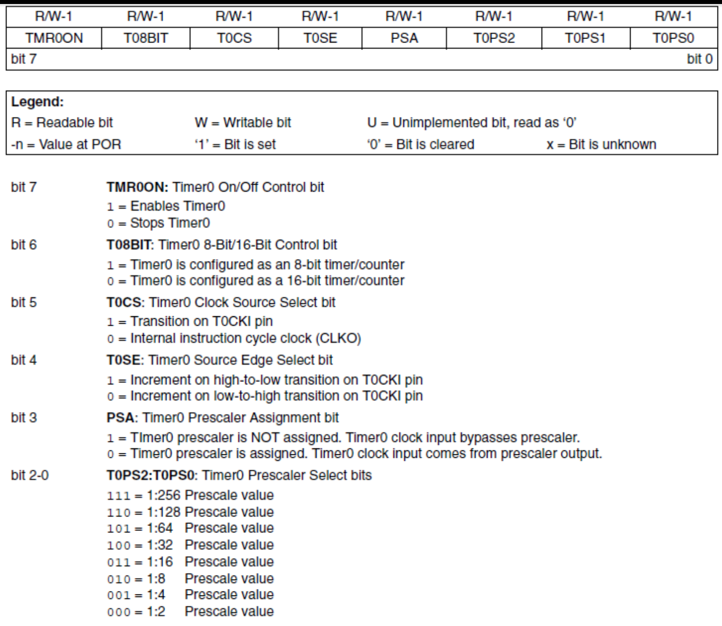


Bit Value	TMR0 Rate
000	1 : 2
001	1 : 4
010	1 : 8
011	1 : 16
100	1 : 32
101	1 : 64
110	1 : 128
111	1 : 256

Figure 3: Timer 0 operation[3]

B. Timer and Interrupt Configuration

The PIC18F has 3 timers which can be used independently of each other. Timer 0 is used in this project. The timer is configured using T0CON. The format of T0CON is as shown in Fig 4 referred from [3]. Different prescalers can be assigned depending on the combination of prescaler bits of the T0CON register. The timer is configured as 8 bit timer with prescaler of 64 and the internal clock used as clock source. The MSB of the register is made 1 so as to enable the timer.



R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-Bit/16-Bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits
111 = 1:256 Prescale value
110 = 1:128 Prescale value
101 = 1:64 Prescale value
100 = 1:32 Prescale value
011 = 1:16 Prescale value
010 = 1:8 Prescale value
001 = 1:4 Prescale value
000 = 1:2 Prescale value

Figure 4: T0CON Register[3]

C. *Frequency Generation*

To generate a signal of any frequency, you need to generate a delay which is inverse of the frequency. To do that, the following steps should be followed.

- Ascertain the oscillator frequency of your controller circuit. If the controller is running on internal oscillator, it operates at 48 Mhz.
- The machine or instruction frequency is one-fourth of this oscillator frequency.
- This means that the time taken for one increment of the timer register is the time taken for one machine cycle to execute.
- Find the time required for your frequency by getting the inverse of the frequency and dividing it by two (as the total delay consists of on-delay as well as off-delay).
- Now divide this required delay by the time it takes your timer register to increment by one.
- The number you get is the number you need to subtract from 255(assuming the timer is configured as 8bit).
- Then this difference is loaded into the timer register and the timer is started.
- This means that the timer will overflow after that period of time generating an accurate delay.
- If the quotient after dividing is more than 255, use an appropriate prescaler to increase the delay.

It is advisable to write a simple program or make an excel sheet with all these calculations to find the load values easily and quickly. Also verify the frequency on a DSO.

V. TONES

The major scale in music follows a fixed ratio between the notes of the scale and the base note of the scale. This pattern is shown in table 1

Note	Ratio w.r.t base note	Frequency
Sa(Base note)	1	261
Re	9/8	293
Ga	5/4	329
Ma	4/3	349
Pa	3/2	392
Dha	5/3	440
Ni	15/8	493
Sa(upper octave)	2	522

Table1. Frequency ratios

As shown in the table, knowing the base note frequency, we can generate the tones for the entire major scale for that note. It can also be seen that the upper octave note for the base note is twice the base note and the next octave note is twice of that showing that these frequencies are logarithmic like our hearing.
The frequencies for each note are shown in fig. 5 for better understanding.

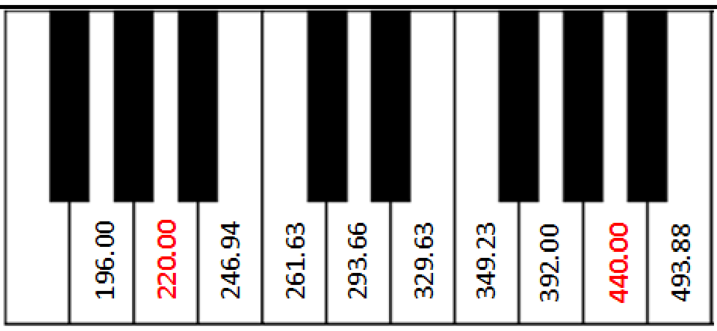


Figure 5: Note Frequencies

Knowing the relation between the base note frequencies and frequencies of other notes, we can have an array of delays that we need to generate. The different values to be loaded into the timer register for each frequency can be independently calculated and stored in an array.

These frequencies can be played in a specific order with appropriate delays to play a certain song.

VI. Results

The frequency generation method was used to generate a sample frequency of 430 Hz as shown in figure 6.

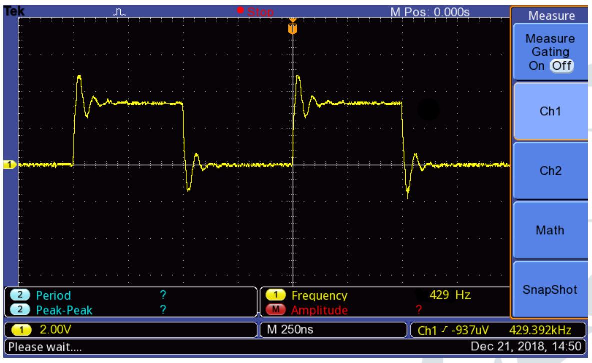


Figure 6: Frequency verification.

This method was used to play the C major scale in series in ascending and descending order.
Some well-known songs were programmed to be played to verify the sequential algorithms.

VII. Conclusion

In this paper, a project for generating various tones using Pic18F microcontroller has been developed. The paper describes the method for frequency generation using Timer 0 peripheral of Pic18F4550 controller. The relation between base note of a scale and the other notes as well as octave notes has also been explored. Using this system, any musical note or tune can be recreated using a microcontroller and a minimal audio setup with reasonable accuracy.

VIII. References

[1] User Manual for Aurum v1.2 Deepak Bhat, Ashwin Radhakrishnan, Debapratim Ghosh, Microcontroller-based Development Team, Wadhwani Electronics Lab, Indian Institute of Technology Bombay.

[2] BC110 Transistor datasheet

[3] PIC18F4550 datasheet, Microchip Inc.