ECE 3720

Microcomputer Interfacing Laboratory

Section 005

Aaron Bruner

Date Performed: 03 / 4 / 2021

Lab 7

**ABSTRACT**:

A lab designed to demonstrate how to program the PIC32 microcontroller to utilize its internal timer functions. When we combine the internal timers with interrupt signals we can then introduce the piezo buzzer. Similar to the previous labs, this will also enhance skills in reading documentations for new and existing parts.

**INTRODUCTION:**

The goal of lab 7 is to program our microcontroller to understand how to program our PIC32MX to sing us a song. We will be setting up the internal timers to trigger an interrupt at frequencies corresponding to musical notes. The interrupt will toggle the digital output to a piezo buzzer, causing it to produce the desired note, and play a song. To perform our lab, we required a variety of materials including:

* A breadboard
* Wires
* Push Button
* Piezo Buzzer
* PIC32MX150F128D
* Analog Discovery 2 (AD2)
* Digilent WaveForms
* MPLAB Snap Debugger
* MPLAB X IDE (Programming Software)

The AD2 will power the PIC32 chip along with partially powering the MPLab Snap Debugger so that it can write to the PIC32. The majority of the power for the MPLAB Snap Debugger will come from the Micro USB cord.

**EXPERIMENTAL PROCEDURES:**

We must assume that the individual reproducing this lab has already setup their breadboard in a manner that their PIC32 chip can be programmed using the MPLAB X IDE software.

To begin the lab, we want to analyze the figure one to see how we need to wire up our circuit. This is most likely the simplest circuit for any lab performed. Since the piezo buzzer is omnidirectional it doesn’t really matter which way you place it. Also, we did not need to use a button, we could have made one side of the piezo buzzer ground and the other connected to pin 16 or B7. However, since the lab instruction asked for a button we provided one.

For the code, the first portion of this code was provided to use in the lab manual. We are first setting up our notes and what values those need to represent so that we can get a sound we recognize. Then, we setup an array of musical notes which will play the desired song we want. After the provided code we begin by developing our interrupt signal which is for our timer. Every time the interrupt is signaled we set the PR2 value to the next musical note in our song and check to see if it’s anything other than r which is our rest sound. If it is r then we do not play anything, otherwise we play that note to the piezo buzzer. We increment j so that we can get to the next beat of the song and then finally reset the interrupt signal flag.

The main function is also not very complex. First, we copied the interrupt setup code from the previous lab. Next, we want to disable our timer so that we can set the options we want. First, we will initialize the prescale value to 0x000 which makes it 1:1. Next, we enable our 32-bit timer since that was an optional part of the lab. This does not make any big difference since we could play the song with either a 16- or 32-bit timer. Lastly, we set the timer to utilize the internal peripheral clock instead of an external clock.

As we continue, the next portions of code are further setting up our timers. We start by clearing the TMR2 register which is the timer we are using for this lab. Then, we load our period register with all 1s.

Now that we have setup our timers we want to enable the timer three for the 32 bits. Similar to the last timer we will clear the interrupt signal and set the polarity level. Then, we set B7 to be an output; this is the output for our piezo buzzer. Then, we set the timer two ON register to one which will enable the timer two.

The while loop was provided to us and we were instructed to not modify anything in the while loop.

**RESULTS and DISCUSSION:**

Once the microcontroller was programmed it was apparent that things were working as expected when we pressed the push button we started to hear music being played. And as expected, when the button was released it would stop playing the song, then when pressed again it will begin to play again.

One problem during this experiment was that the piezo buzzer would not making any noise while the program was running and the push button was pressed. It turned out that the piezo buzzer was actually not functional and simply replacing the buzzer fixed the issue.

This week we demonstrated how to utilize the idea of timers and further knowledge of interrupts. Being able to read the datasheet and determine what pin everything needs to be connected to and how to properly setup your timer is crucial in the real world. Managing multiple timers and combining them with interrupts will be very common in microcontroller programming careers.

**CONCLUSION:**

To conclude, lab 7 taught us many useful techniques on handling internal timers and combining those with interrupts on the PIC32 microcontroller. Also, as with many other labs, this one teaches us how to debug our circuit to check if everything is working. This lab also **Diagram, schematic

Description automatically generated**taught us more complex ways to analyze problems and how to solve them.

**FIGURES AND TABLES:**

**Figure 1: Wiring for lab 6 (Pin connections described in experimental procedures)**

**CODE:**

#include <xc.h>

#include <sys/attribs.h>

//This is here to make timing calculations easier!

#pragma config FPBDIV = DIV\_2

//r is a Rest. If r, do not toggle! (don't want noise to play)

#define r 3000

#define a 4545

#define b 4050

#define C 3817

#define C\_ 3610

#define D 3402

#define D\_ 3216

#define E 3031

#define F 2866

#define F\_ 2703

#define G 2551

#define G\_ 2410

#define A 2273

#define A\_ 2146

#define B 2025

#define CC 1911

#define q 400

#define qdot q \* 1.5

#define e q/2

#define s e/2

#define t32 s/2

#define sdot s+t32

#define h q\*2

#define hdot q+e

#define edot e+s

#define num\_notes 52

int i,j;

short delay[num\_notes] = {t32,t32,t32,t32,t32,t32,t32,t32,t32,t32,s,sdot,t32,t32,

t32,t32,t32,t32,t32,t32,t32,t32,t32,s,sdot,t32,t32,t32,

t32,t32,t32,t32,t32,t32,t32,t32,s,sdot,t32,t32,t32,t32,

t32,t32,t32,t32,t32,t32,t32,t32,s,e};

short music\_notes[num\_notes] = {b,r,B,r,F\_,r,D\_,r,B,F,r,D,r,C,r,CC,r,G,r,E,r,CC,G,r,

E,r,b,r,B,r,F\_,r,D\_,r,B,F,r,D,r,D,E,D\_,r,F,F\_,G,r,G,G\_,A,r,B};

void \_\_ISR(12)Int0Handler(void){

PR2 = music\_notes[i];

(music\_notes[i] == r) ? : (LATBbits.LATB7 ^= 1); // If rest skip, else play

j = j + 1;

IFS0bits.T3IF = 0; // Clear interrupt flag

}

int main(void) {

INTCONbits.MVEC = 1;

\_\_builtin\_enable\_interrupts();

CFGCONbits.JTAGEN = 0;

T2CONbits.ON = 0; // Disable Timer2

T2CONbits.TCKPS = 0x000; // Utilize a prescale value of 1:1

T2CONbits.T32 = 1; // 32-Bit Timer

T2CONbits.TCS = 0; // Internal peripheral clock

TMR2 = 0x00; //Clear contents of the timer register

PR2 = 0xFFFF; // Load the Period register with the value 0xFFFF

IEC0bits.T3IE = 1; // Enable Timer3 interrupts

IFS0bits.T3IF = 0; // Clear the Timer3 interrupt status flag

IPC3bits.T3IP = 1; // Polarity level for Timer3

TRISBbits.TRISB7 = 0; // B7 Output

T2CONbits.ON = 1; // Enable Timer2

while (1)

{

for (i = 0; i < num\_notes; i++) {

j = 0;

while (j < delay[i]) { //length of time to play note

}

}

}

}