Johan Ospina

EC 450

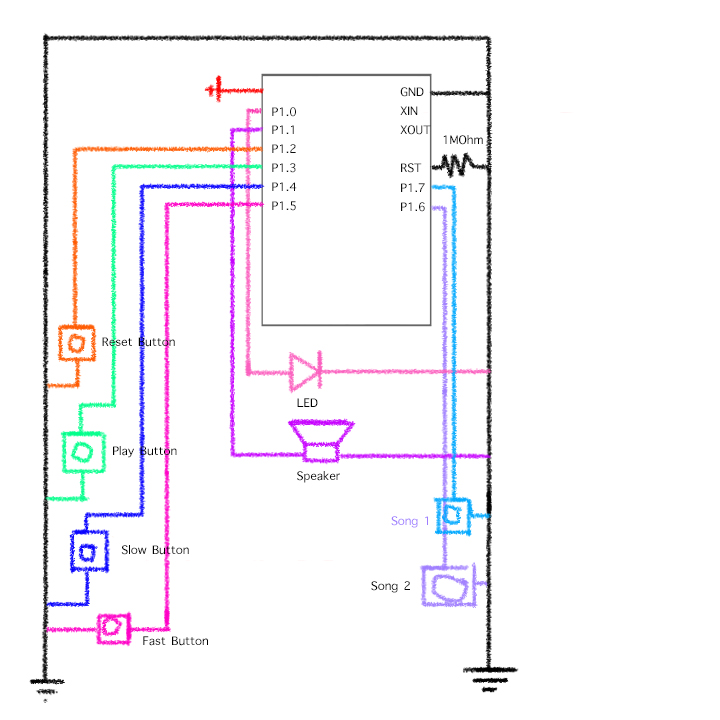
Homework 3

3/17/2016

**Setup**

The device consists of a speaker, 6 buttons, an LED, and multiple wires and resistors.

(Figure 1 Schematic of implementation)



The speaker is used by Port1 pin P1.1 using the Timer A Capture Compare register output as its frequency source. Interaction is facilitated through the six buttons on the breadboard. The Microcontroller was placed onto the breadboard and a pull up resistor was placed on the RST pin to allow for operation. A voltage of 3V was applied to the Vcc pin and then it was attached to GND on the bread board.

|  |  |
| --- | --- |
| P1.0 | Indicator LED |
| P1.1 | Speaker |
| P1.2 | Reset Btn |
| P1.3 | Play/Pause Btn |
| P1.4 | Slower Btn |
| P1.5 | Faster Btn |
| P1.6 | Song 1 Btn |
| P1.7 | Song 2 Btn |

**P1.0** LED indicates various system states (ready/paused/playing.

**P1.1** Speaker is connected to ground

**P1.2** Reset button restarts the current song

**P1.3** Plays the song that is currently selected

**P1.4** Button slows down the song while it’s playing

**P1.5** Button speeds up the song while it’s playing

**P1.6** Selects song 1 –Joy To The World-

**P1.7** Selects song 2 –We bare bears & Steven Universe themes

As it is shown Port one is completely filled by the various apparatuses that are attached to it.

**Code Explication:**

Each song I created is stored inside of a byte array to minimize space. The closes C has for bytes are character arrays and therefore those were used to ensure maximum compactness. Through this implementation the index of the note frequency is held within the last five bits of the song element and the index of the duration for the note was held within the first 3 bits of the array.

Ex: A sixteenth note F5# note was represented by the byte

[001 10011]

This way we can logically shift the byte by 5 to the right to get the index of the duration array to query for how many WDT interrupt cycles we wanted to play our note for (in this case it is durations[1] == 8 WDT interrupts. After we have acquired our duration index we could then apply a bit mask onto the byte (000 11111) to isolate the value of the index for the frequency array that we would like to query. Again, for this case we would get an index value of freq[19] == 676.

I have stored the frequencies as half periods using the relationship

This is due to the fact that at the beginning of the program I use the Basic Clock System control register and Digitally Controlled Oscillator register to set the frequency of SMCLK to 1Mhz. Then I set the CCR0 register to that half period in order to generate a square wave with the correct frequency.

For easy writing there are a bunch of definitions in the code for both durations and also frequencies that can be used to index into the array that we want to read from.

**Song Playing:**

When one of the two songs is chosen for playback two variables called WDT\_duration and score\_counter are set to zero. WDT\_duration is used to measure how many watch dog timer interrupts occur as a note plays and score\_counter is the index of the current song we are playing. Once a WDT\_duration value goes from zero to the value corresponding to the first three bits of the byte we are looking at, the score\_counter index will increment by one to move on to the next duration-frequency pair. WDT\_duration is computed by multiplying the default value of a note with a tempo variable. This allows me to change the tempo of the song by just increasing or decreasing the tempo multiplier, as it will affect all of the notes equally.

For example:

A quarter note is by default 2.25 \* 32 WDT interrupts long, which is about half a second. If we changed the value of the tempo multiplier from 2.25 to something else, this duration could either increase or decrease and thus play the song faster or slower.

**Virtues and Limitations:**

The downside of this implementation is how little space I have to represent duration intervals. This is due to the trade off with C because I can’t use a half byte extra without just adding a whole byte. I found that just using one byte was effective for most music, however, and allowed me to play most songs with little modifications. I did run into trouble with the Steven Universe opening, since it used a double dotted half note and I did not have this as a readily representable note.

Also the button interrupts being handled is directly dependent on the timing for the watch dog interrupt handler, If I press the buttons while that is being handled I will not be able to change my state values.

On the bright side, the way the notes get stored is very compact as it only takes a byte, so the size of my music is dependent only on the size of the free bytes in memory. It also is able to efficiently change states and vary the tempo while not using that much computational power (like if I were polling the buttons for example).