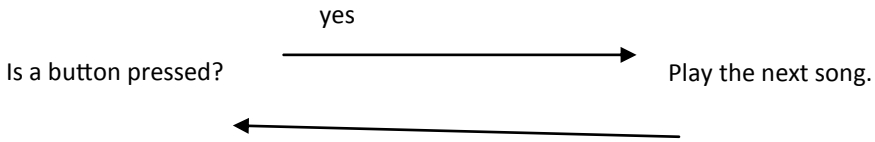


```
/*****************************************************************
 * Title: Lab 7 Assignment
 * Date Due: Oct sometime
 *
 * Description:
 * Here we will use Timer_B to modulate the output of Port 3.5 (TB4)
 * in order to drive an annoying-as-fuck speaker on the Exp Board!
 * The main idea is that you /must/ run both cap/comp 0 and 4. Cap/comp
 * 0 is used to drive the clock and TBCCR0 sets the max value for most
 * of the MCx settings, such as "UP" mode.
 * In order to make the calculations easier, I'm going to use 'toggle'
 * mode in this program, which means that TBCCR4 *doesn't matter* and
 * we can module the speaker directly with TBCCR0 :D
 * Also, we use watchdog for something, I think...
 *
 * By: Christopher 'ctag' Bero <csb0019@uah.edu>
 * Project: https://github.com/ctag/uah
 *
 * This software is licensed under the GPLv3,
 * consult the LICENSE file for more information.
 *****************************************************************/

#include <msp430.h> // The *correct* include statement
#include <math.h> // for pow(), puh power!

#define SW1 (0x01&P1IN) // B1 - P1.0 switch SW1
#define SW2 (0x02&P1IN) // B2 - P1.1 switch SW2
```

```
#define LED1 BIT2; // LED1 - P2.2 - 0x04
#define LED2 BIT1; // LED2 - P2.1 - 0x02

// Usage: WDTCTL = WDT_CONFIG;
#define WDT_CONFIG_250 (WDTPW|WDTCNTCL|WDTSSEL|WDTIS0) // Set bits to give us 0.250s watchdog
#define WDT_CONFIG_1000 (WDTPW|WDTCNTCL|WDTSSEL) // Set bits to give us 1s watchdog
#define WDT_INTERVAL_250 (WDTPW|WDTCNTCL|WDTTMSEL|WDTSSEL|WDTIS0) // Set bits to have 0.250s timer
#define WDT_INTERVAL_1000 (WDTPW|WDTCNTCL|WDTTMSEL|WDTSSEL) // Set bits to have 1s timer
#define WDT_HALT (WDTPW|WDTHOLD) // Set bits to halt the timer

short unsigned int new_note_flag = 0;

double freq[] = {16.35, 17.32, 18.35, 19.45, 20.60, 21.83, 23.12, 24.50, 25.96, 27.50, 29.14, 30.87, 1.00};
char notes[] = {'C', 'd', 'D', 'e', 'E', 'F', 'g', 'G', 'a', 'A', 'b', 'B', ' '};

// P = Pulse
// N = No Pulse

char songs[3][66] =
{
/*Twinkle Twinkle, Little Star*/
        'P', /*Pulse*/
        'C','C','G','G','A','A','G','G','G','G',' ',' ','F','F','E','E',
        'D','D','C','C','C','C',' ',' ',
        'C','C','G','G','A','A','G','G','G','G',' ',' ','F','F','E','E',
        'D','D','C','C','C','C',' ',' ',
        ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
        'Z', /*FIN*/

/*Jolly Old St. Nicholas*/
        'N', /*No Pulse*/
        'B','B','B','B','A','A','A',' ','G','G','G','G','B',' ',' ',' ',' ',
        'E','E','E','E','D','D','G',' ','A','G','A','B','A',' ',' ',' ',' ',
        'B','B','B','B','A','A','A',' ','G','G','G','G','B',' ',' ',' ',' ',
        'E','E','E','E','D','D','G',' ','A','G','A','B','G',' ',' ',' ',' ',
```

```
/*We Three Kings*/
        'N', /*No Pulse*/
        'B',' ','A','G',' ','E','g','G','g','E',' ',' ',
        'B',' ','A','G',' ','E','g','G','g','E',' ',' ',
        'G',' ','G','A',' ','A','B',' ','B','D','C','B',
        'A','B','A','G',' ','g','E',' ',' ',' ',' ',' ',' ',
        ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
        ' ',' ',' ',' ',' ',
        'Z' /*FIN*/
};

short unsigned int songs_octive[3][66] =
{
/*Twinkle Twinkle, Little Star*/
        0,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
        -1, /*FIN*/

/*Jolly Old St. Nicholas*/
        0,
        5,5,5,5,5,5,5,0,5,5,5,5,5,0,0,0,
        4,4,4,4,4,4,5,0,5,5,5,5,5,0,0,0,
        5,5,5,5,5,5,5,0,5,5,5,5,5,0,0,0,
        4,4,4,4,4,4,5,0,5,5,5,5,5,0,0,0,
        -1, /*FIN*/

/*We Three Kings*/
        0,
        5,0,5,5,0,4,5,5,5,4,0,0,
        5,0,5,5,0,4,5,5,5,4,0,0,
        5,0,5,5,0,5,5,0,5,6,6,5,
        5,5,5,5,0,5,4,0,0,0,0,0,
```

```
        0,0,0,0,0,0,0,0,0,0,0,0,
        0,0,0,0,
        -1 /*FIN*/
};

int index = 1;

int songs_num = 2;
int current_song = 0;

int rest = 0;
int pulse = 0;
char note = ' ';
short unsigned int octive = 0;

void main(void)
{
        WDTCTL = WDT_HALT;

        P1IE |= 0x0003;             // P1.0 interrupt enabled
        P1IES |= 0x0003;  // P1.0 hi -> low edge
        P1IFG &= ~0x0003;           // Clear P1.0 IFG

        IE1 |= BIT0; // BIT0 is WDTIE
        __enable_interrupt(); // Enable global interrupts

        P3DIR |= BIT5; // Set P3.5 to OUT
        P3SEL |= BIT5; // Setup the special TB4 function of P3.5

        /* Setup Timer_B's TBCTL
         * BITs Mapping for TxCTLx:
         *      [15]    [14]    [13]    [12]    [11]    [10]    [9]             [8]
         *      NA              NA              NA              NA              NA              NA              TASSEL
        TASSEL
         *      0               0               0               0               0               0               0
                1
```

```
*       0       0       0       0       0       0       0
        1
*
*       [7]     [6]     [5]     [4]     [3]     [2]     [1]
        [0]
*       ID      ID      MC      MC      NA      TACLR   TAIE    TAIFG
*       0       1       0       1       0       0       0
        0
*
* Condensed: [0000][0001] [0101][0000]
*
* Specifics:
* TASSELx = 01, ACLK
* IDx = 01, divide ACLK by 2, ACLK/2 = 16,384
* MCx = 01, UP mode
*/
TBCTL = 0x0150;

/* Choosing a value for TBCCR0:
 * This is a little tricky, because you can tackle this problem
 * from a multitude of angles, and the one I chose is likely non-standard,
 * but makes much more sense to me.
 * So, TBCCR0 represents the period out of 16,384 which we wish
 * for the speaker to be pulsed each second. A value of 16,384 means
 * the speaker will click once per second. A value of 16 means the
 * speaker will click about 1000 times per second, which is an audible tone!
 *
 * The formula:
 * TBCCR0 = [ACLK = 16384] / [desired freq in Hz]
 *
 * So, for 900Hz, TBCCR0 = [16384] / [900] = 18.2 = 18.
 */
 //TBCCR0 = 18;

/* Setup Timer_B's TBCCTL4
 * BITs Mapping for TxCTLx:
```

```
*       [15]    [14]    [13]    [12]    [11]    [10]    [9]             [8]
*       CM              CM              CCIS    CCIS    SCS             CCLD    CCLD    CAP
*       0               0               0               0               0               0               0
        0
*
*       [7]             [6]             [5]             [4]             [3]             [2]             [1]
        [0]
*       OUTMOD          OUTMOD          OUTMOD          CCIE    CCI             OUT             COV
CCIFG
*       1               0               0               0               0               0               0
        0
*
* Condensed: [0000][0000] [1000][0000]
*
* Specifics:
* OUTx = 100, Toggle mode (on TB4 of course)
*/
TBCCTL4 = 0x0080;

TBCCR4 = 1; // doesn't matter, can be any valid value

while (1)
{
        if (new_note_flag == 0)
        {
                //LPM3; // Turn off mclk and smclk
        }

        new_note_flag = 0;

        for (int i = 0; i < 13; i++)
        {
                if (notes[i] == note)
                {
                        if (notes[i] == ' ')
                        {
```

```
                                        TBCCR0 = 0; // halt the buzzer for a rest
                                } else {
                                        TBCCR0 = (16384 / (int)(freq[i] * pow(2,octive) ) ); // Set the correct period to achieve a
note
                                        break; // break from for loop
                                }
                        }
                }
        }
}

/*
 * Watchdog interrupt service
 */
#pragma vector = WDT_VECTOR
__interrupt void blink_watchdog(void)
{
        new_note_flag = 1;
        if (rest == 0)
        {
                index ++;
                note = songs[current_song][index];
                octive = songs_octive[current_song][index];

                if (songs[current_song][index] == 'Z')
                {
                        index = 1;
                        WDTCTL = WDT_HALT;
                } else if (pulse == 1) {
                        rest++;
                }
        } else {
                note = ' ';
                octive = 0;
                rest = 0;
```

```
        }
}

/*
 * Port 1 interrupt service
 */
#pragma vector = PORT1_VECTOR
__interrupt void Port1_ISR (void)
{
        current_song++;

        if (current_song == 3)
        {
                current_song = 0;
        }

        if (songs[current_song][0] == 'P')
        {
                pulse = 1;
        } else {
                pulse = 0;
        }

        WDTCTL = WDT_INTERVAL_250;

        P1IFG &= ~0x0003;           // Clear P1.0 IFG
}
```