Lab #2

MSP430 Hero! Digital IO Meets Timers & Interrupts

ECE2049 - Embedded Computing In Engineering Design

Filippo Marcantoni & Anthony Previte

02/08/2023

Introduction:

In this lab we created our second game using the MSP430F5529 Launchpad-based Lab Board resembling the Guitar Hero game. Our MSP430 Hero game version gives a menu of two songs and after one has been selected by pressing 1 or 2 on the keypad, it starts playing the song. The song is dependent on a timer, in which at the sound of each note a LED flashes indicating the note that the player should press using the corresponding push buttons on the board. During the game, the player receives feedback from the two Launchpad User LEDs, the red LED1 blinks when the button pressed doesn't correspond to the right note and the green LED2 blinks if they match. The game can be reset to the Welcome screen during the game when the key '#' is pressed on the keyboard.

Discussion and results:

Pre Lab) For the pre lab we had to write a number of functions. The first one (setup) configured the four lab board buttons. The next one (pressed) returned the state of the board buttons with 1 being on and 0 being off in the form of 000000000 where the final four digits correspond to the buttons, so button 4 and 3 being on would return 00001100. Next we configured the LED's and wrote a function to determine when they're on, taking an argument similar to when the buttons are pressed as previously described. We were able to map the pitch, LED and duration to each note creating a structure called Notes with the respective fields. Functions can be seen below:

```
228 void setup(){
229     P7SEL &= ~ (BIT4|BIT0);   //bit 4 and 0 are 0
230     P3SEL &= ~ (BIT6);   //bit 6 is 0
231     P2SEL &= ~ (BIT2);   //bit 2 is 0
232
233     P7DIR &= ~ (BIT4|BIT0);
234     P3DIR &= ~ (BIT6);
235     P2DIR &= ~ (BIT2);
236
237     P7REN |= (BIT4|BIT0);
238     P3REN |= (BIT6);
239     P2REN |= (BIT2);
240
241     P7OUT |= (BIT4|BIT0);
242     P3OUT |= (BIT6);
243     P2OUT |= (BIT2);
244 }
245
```

```
246 int pressed(){
247     int on = 0x00;
248     int S1 = P7IN & BIT0;
249     int S2 = P3IN & BIT6;
250     int S3 = P2IN & BIT2;
251     int S4 = P7IN & BIT4;
252
253     if (S1 == 0){
254         on = (on|BIT3);
255     }
256     if (S2 == 0){
257             on = (on|BIT2);
258         }
259     if (S3 == 0){
260             on = (on|BIT1);
261         }
262     if (S4 == 0){
263             on = (on|BIT0);
264         }
265     return on;
266 }
267
```

```
268 void configUserLED(char inbits){
269     char mask = 0;
270
271     if (inbits && BIT0){
272         mask |= BIT4;
273     }
274
275     if (inbits && BIT1){
276         mask |= BIT3;
277     }
278
279     if (inbits && BIT2){
280         mask |= BIT1;
281     }
282
283     if (inbits && BIT3){
284         mask |= BIT2;
285     }
286     P6OUT|= mask;
287 }
```

1) We created our welcome screen using graphics drawing functions in our case 0. When starting the game, a Welcome to MSP430 Hero message is displayed on the LCD and gives the option of either pressing the key '*' to start the game, or the key '#' to reset it. The game can be reset at any time except during the countdown.

2) Once the key '*' is pressed, a menu presenting two songs, Smoke on the Water and Three Blind Mice, is displayed. At this point the player can choose one of the two songs by pressing the key '1' or '2' on the keypad, choosing a song. The LCD then displays a countdown, implemented with our Timer A2, for which a LED flashes at each number and, when GO is displayed, all of the board LEDs flash with a buzzer sound to signal the game is starting.

   In the final version, the count down must be measured by Timer A2 and NOT implemented using software delays. Explain the difference between event (or interrupt) driven code and polling. Is your final code strictly event driven or will you use a mix of interrupts and polling? Explain in your report.

   Event driven code uses a clock whereas polling depends on the operating system and timing intervals can change based on a number of factors. In order for the countdown to represent the correct amount of time, interrupts must be used. We can set this amount of time to be whatever we want this way.
   Our final code is event driven and no longer has software delays present.

3) The only modification we made to the buzzer on function from peripherals was adding a parameter of frequency and dividing 32768 by it to obtain the desired period. Doing so, we were able to set the PWM period to the correct amount of ACLK tics to produce the right pitch for the given note.
   You will need to do some math to convert these frequencies to the number of ACLK tics. Discuss your conversion of frequency in Hz to Timer B CCR0 settings in your report

In order to get the right pitch for a given note, in TB0CCR0 we set the PWM period to ACLK tics, which we found by dividing 32768 (ACLK frequency) by the desired frequency of each note. Then after giving a pitch (note's frequency) to each note, we call the modified buzzer function every time a note is played. with each note having its respective frequency were able to produce that note with our function. The function can be found below.

```
void BuzzerOn2(int freq)
{
    // Initialize PWM output on P3.5, which corresponds to TB0.5
    P3SEL |= BIT5; // Select peripheral output mode for P3.5
    P3DIR |= BIT5;

    TB0CTL  = (TBSSEL__ACLK|ID__1|MC__UP);  // Configure Timer B0 to use ACLK, divide by 1, up mode
    TB0CTL  &= ~TBIE;                       // Explicitly Disable timer interrupts for safety

    // Now configure the timer period, which controls the PWM period
    // Doing this with a hard coded values is NOT the best method
    // We do it here only as an example. You will fix this in Lab 2.
    int rate = 32768/freq;
    TB0CCR0   = rate;                   // Set the PWM period in ACLK ticks
    TB0CCTL0 &= ~CCIE;                  // Disable timer interrupts

    // Configure CC register 5, which is connected to our PWM pin TB0.5
    TB0CCTL5  = OUTMOD_7;                   // Set/reset mode for PWM
    TB0CCTL5 &= ~CCIE;                      // Disable capture/compare interrupts
    TB0CCR5   = TB0CCR0/2;                  // Configure a 50% duty cycle
}
```

4) We created a struct for our notes, in which each note has three fields: a duration, an assigned LED mask and a frequency. We made a new type of note for each lettered note so there were many notes to choose from in our songs. We then created an array of 28 notes long for each song that could be played when the game begins.

   How will you control the duration of your notes? Will you do this within the buzzer function or within the main game loop? Remember you will need to be checking buttons during the notes. Explain your choice in your report.

   We controlled the duration of our notes within our note struct in the member called duration, and in our play note function. If we wanted a note to have a different duration, we would create a new note that had the same pitch and LED, but different duration. The duration was an integer within the note struct containing the number of ACLK tics it would endure. In our play note function we are controlling the duration of the notes with a while loop, which plays the note frequency using our modified buzzer funcion, which runs when the difference between the timer count and the time that has passed in the song is smaller than the note duration. Doing so, we know that the buzzer function will sound only in the interval for which the note is supposed to play.

5) **Explain in your report why software delay would then no longer work and why you must implement note duration using the timer interrupts.**

   Software delays don't allow simultaneous checking of buttons, only at the end or beginning of the delay. In contrast, timer interrupts can be used to check for button

pressed inputs multiple times per second based on the resolution. We implemented note duration using timer interrupts as well so that the time would be accurate and so that the timer would have a range of time to be checking for user inputs while the notes are being played.

6) **Explain in your report how you setup Timer A2 and why Timer A2's resolution should be several times smaller than the duration of a note**

   We set up Timer A2 with our function startTimerA2(void), which starts the timer count when called and sets the Auxiliar Clock (ACLK) as clock source. It also sets it to up mode and the divider to be 1. Then after setting our max count in TA2CCR0 to be 163, to have a resolution of 0.005 seconds, we enabled Timer A2 interrupt. We can stop Timer A2 calling our function stopTimerA2(void) which disables Timer A2 interrupt and sets the timer count back to 0. We finally created our Interrupt Service Routine which includes a leap count, since our maximum count was 162.84 and not exactly 163, to check and fix the count when the timer is off by 5 ms, which happens after 1024 interrupts. Since the maximum count is actually smaller, our timer, instead of running at precisely 5 ms, runs at 5.0048828 ms and perhaps our timer is actually slower. We set the note duration to be 100 ACLK tics, which correspond to half of a second. We use the timer so the note duration can be set to any duration accurately too. The resolution is several times smaller than the duration of a note so while the note is playing the interrupts can seemingly constantly be checking for buttons.

7) **Explain your rules for scoring and losing and how you implemented them in your report.**

   Our rules were very simple. Each correct note gained a point and you had to achieve a total of at least 18 points out of 28 to win, anything less counted as a loss. An incorrect note resulted in no change in the points. We did this in our play song function, in which we are checking if the button pressed correspond to the right LED, which count the points while the song is being played and returns the number of points scored once the song finishes or when the ‘#’ key is pressed. We then used an if statement in our case 1 to determine if the number of points was less than or greater than 18 and printed ‘WINNER’ or ‘LOSER’ respectively on the LCD.

Conclusion:

In doing this lab we gained a deeper understanding of timers as well as their usefulness and applications. We successfully created a MSP430 Hero game that played a song and the player pressed buttons to match LEDs when they lit up that correspond to the note of the song being played at the time. We were able to configure and use the 4 LEDs on the board and the 2 LEDs on the Launchpad. We also were able to configure and use the 4 board buttons as player inputs and check and update LEDs in real time using the timer based on the button's state of being pressed or not.

Appendices:

ASCII Table: https://www.ascii-code.com/
MSP430F5529 Schematics: https://canvas.wpi.edu/courses/44793/pages/schematics