Subsystem Design

Digital to Analog Converter to Piezo Buzzer

1 Description

This subsystem demonstrates how to create musical tones on a Piezo buzzer through the use of the digital to analog converter (DAC) of the MSPM0. This process uses predefined frequencies being output to a Piezo to create a tune or jingle, being useful in applications such as home appliance, industrial machinery, or personal electronics.

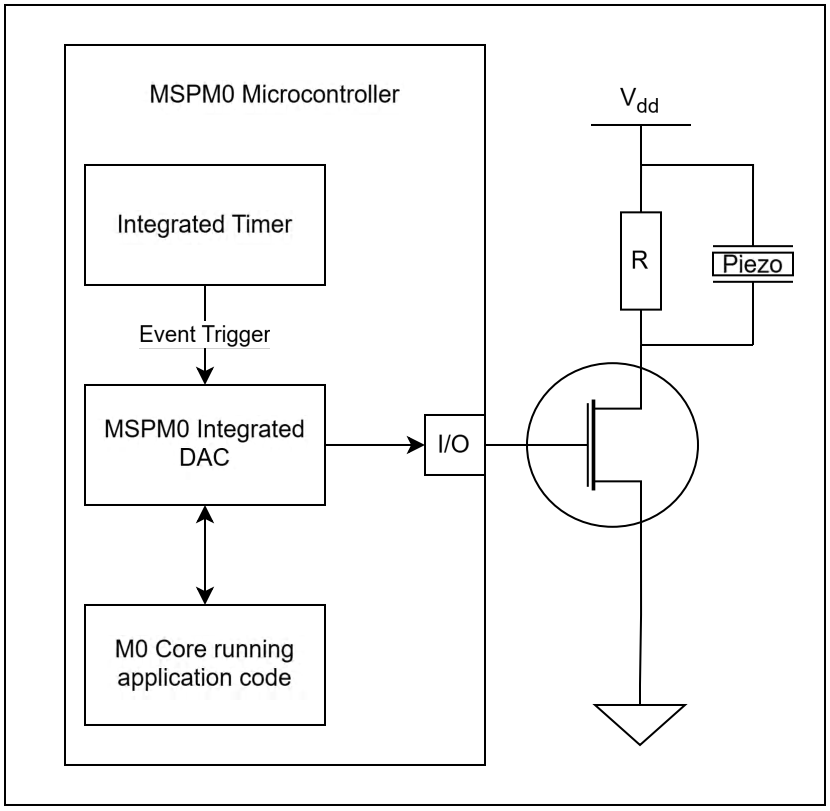

Figure 1-1. DAC to Buzzer Subsystem Block Diagram

2 Required Peripherals

Table 2-1. Required Peripherals

Sub-block Functionality	Peripheral Use	Notes
Tone Generator	DAC (DAC0)	12-bit DAC outputs sine wave samples from 64 point lookup table, generating melodic tones using phase accumulator with 24 fractional bits for frequency generation.
Timing and Sample Rate Control	TimerG (TIMER_0_INST)	Timer triggers DAC based on sample rate (16.39 in the example code to accommodate for 61us period). Timer events trigger DAC FIFO updates.
Melody Sequencing	DAC ISR (FIFO_1_2_EMPTY)	State machine ran through the ISR advances through user-defined melody.
Power Management	SYSCTL	Sleep-on-exit enabled for low power operation. Device is only in RUN during startup and ISR execution.

3 Compatible Devices

This subsystem was designed using the LP-MSPM0G3507. All MSPM0 devices which meet the features as described by the required peripherals sections are compatible, albeit with changes to the timers, pin selection, and frequency.

Notably, depending on the device used, the API wrapper for DAC_12 can differ, and as such requires slight edits to the ISR, although the functionality won't change.

4 Design Steps

1. Based on the piezo buzzer being implemented, determine the audio parameters, including the sample rate, waveform type, lookup table resolution, and DAC resolution.
2. Define the frequency of which notes are to be played (for example, NOTE_C4 = 262Hz), organize them into a melody, and define the duration of each note in milliseconds.
3. Calculate the timer period to generate the desired sample rate trigger events. For a 10kHz sample rate, the timer must trigger at a frequency of 20kHz.
4. Generate a sine wave lookup table by pre-computing 256 sine values scaled to the onboard DAC 12 bit range (0-4095), with the midpoint at 2048 for proper AC coupling to the buzzer.
5. In SysConfig, configure the timer as an event publisher to trigger the DAC12 module, set the DAC to operate in FIFO mode with interrupts enabled for half-empty events, and route the DAC output to the appropriate GPIO pin connected to the buzzer as displayed in Figure 1-1.
6. Write application code to initialize the sine table, implement the melody state machine which tracks the current note and remaining notes to be played, implement the DAC interrupt handler that outputs sine values and advances the melody sequence, and finally enable sleep-on-exit mode for low-power operation between samples.

5 Design Considerations

- **Piezo and Tone Frequency Selection:** The code must be edited to cater to the attributes of the piezo buzzer attached to the circuit. The piezo buzzer used to design and configure the provided code is the PS1230P02BT. The sample rate, as well as the frequency of the tones themselves must be altered to the frequency characteristics of any piezo.
- **Sine Table Generation:** In the example code, a basic 64 entry sine table is used for creating the output on the DAC. However this creates a lower quality output, and thus can be scaled to a higher resolution to have a clearer output. The recommendation is that the sine table is either a predefined array of constants, or generated once upon startup using the MathACL module.
- **Sample Rate and Timer:** The sample rate of this subsystem, declared in the code as gSampleRate, must be equivalent to the period of the timer module. For example, if the desired gSampleRate is 20kHz, the timer period must be 0.05ms. Increasing the sample rate can provide better audio quality and smoother sine wave reproduction.

6 Software Flowchart

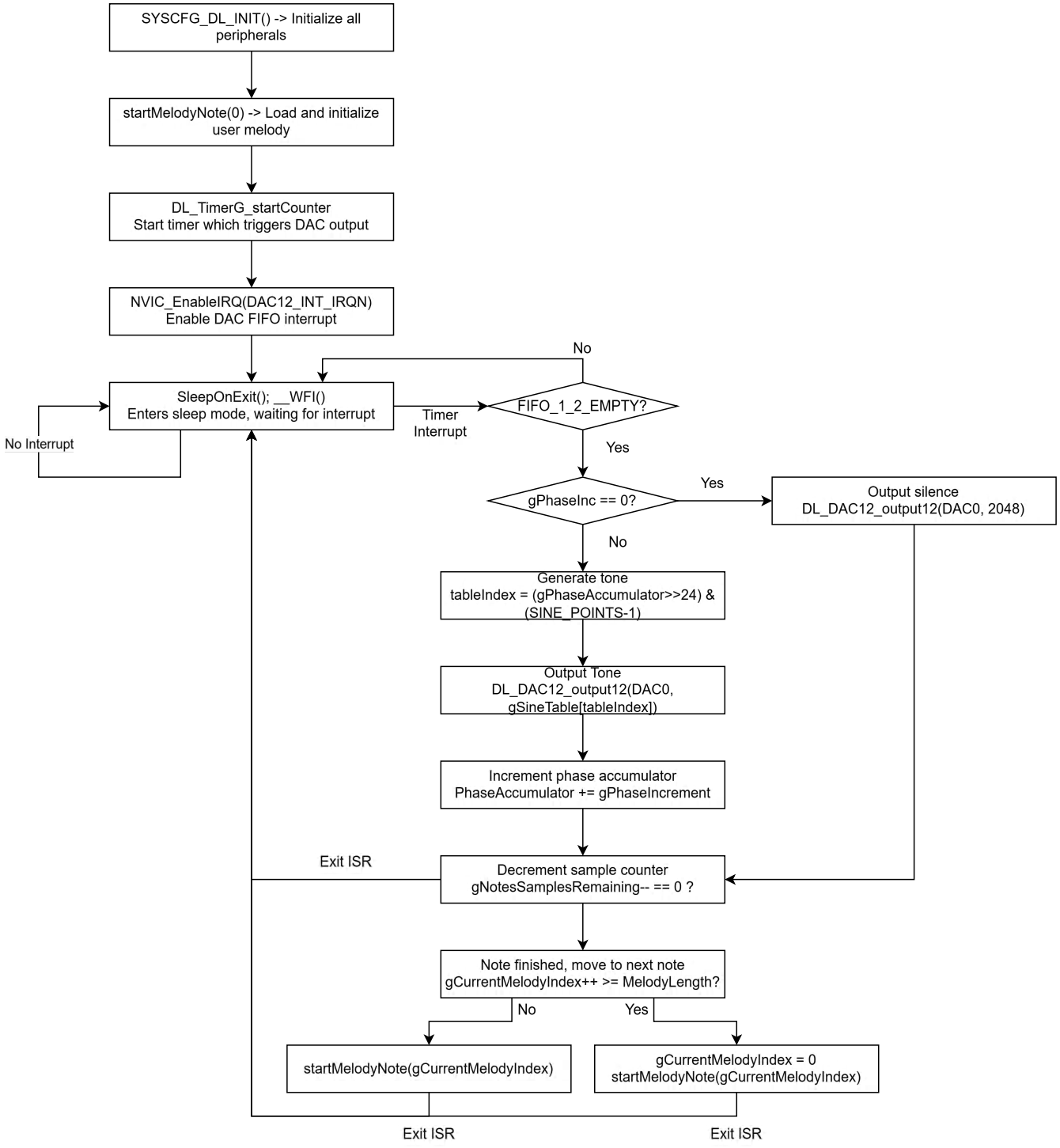

Figure 6-1. Software Flowchart

7 Application Code

```
#include "ti_msp_dl_config.h"

/* musical notes frequencies in Hz - 6th octave */
#define NOTE_C  1047   // C6
#define NOTE_D  1175   // D6
#define NOTE_E  1319   // E6
#define NOTE_F  1397   // F6
#define NOTE_G  1568   // G6
#define NOTE_A  1760   // A6
#define NOTE_B  1976   // B6

/* sine wave lookup table for DAC output */
#define SINE_POINTS 64
const uint16_t gSineTable[] = {2048, 2248, 2447, 2642, 2831, 3013,
    3185, 3347, 3496, 3631, 3750, 3854, 3940, 4007, 4056, 4086, 4095, 4086,
    4056, 4007, 3940, 3854, 3750, 3631, 3496, 3347, 3185, 3013, 2831, 2642,
    2447, 2248, 2048, 1847, 1648, 1453, 1264, 1082, 910, 748, 599, 464, 345,
    241, 155, 88, 39, 9, 0, 9, 39, 88, 155, 241, 345, 464, 599, 748, 910, 1082,
    1264, 1453, 1648, 1847};

/* Variables for tone generation */
#define PHASE_FRAC_BITS 24
uint32_t gPhaseAccumulator = 0;
uint32_t gPhaseIncrement = 0;
/* 16.39kHz sample rate (61us timer period). May require some shifts depending on the piezo used */
uint32_t gSampleRate = 16393;

/* Melody sequence: "BAG BAG GGGGAAAA BAG", "Hot crossed buns" */
/* 0u represents a rest, or a tone of "0" */
static const uint32_t gMelodyFreqs[] = {
    NOTE_B, 0u, NOTE_A, 0u, NOTE_G, 0u, 0u,
    NOTE_B, 0u, NOTE_A, 0u, NOTE_G, 0u, 0u,
    NOTE_G, 0u, NOTE_G, 0u, NOTE_G, 0u, NOTE_G, 0u,
    NOTE_A, 0u, NOTE_A, 0u, NOTE_A, 0u, NOTE_A, 0u,
    NOTE_B, 0u, NOTE_A, 0u, NOTE_G, 0u
};

/* durations in milliseconds for each entry above:
   full note -> 500ms, half note -> 250ms, short rest -> 80ms, phrase separator rest -> 250ms */
static const uint16_t gMelodyDurMs[] = {
    /* One phrase per line */
    500,  80, 500,  80, 500,  80, 250,
    500,  80, 500,  80, 500,  80, 250,
    250,  80, 250,  80, 250,  80, 250,  80,
    250,  80, 250,  80, 250,  80, 250,  80,
    500,  80, 500,  80, 500, 2000
};

/* defines for melody state machine*/
#define MELODY_LENGTH (sizeof(gMelodyFreqs)/sizeof(gMelodyFreqs[0]))
static uint16_t gCurrentMelodyIndex = 0;
static uint32_t gNoteSamplesRemaining = 0;

/* set the phase increment for a given frequency */
void setToneFrequency(uint32_t frequency) {
    if (frequency == 0) {
        gPhaseIncrement = 0;
        return;
    }
    /* compute increment with 24 fractional bits: inc = freq * SINE_POINTS * 2^PHASE_FRAC_BITS /
sampleRate */
    /* allows for smooth frequency shifting */
    uint64_t inc = (uint64_t)frequency * (uint64_t)SINE_POINTS * ((uint64_t)1 << PHASE_FRAC_BITS);
    inc /= gSampleRate;
    gPhaseIncrement = (uint32_t)inc;
}

/* play each note from the melody based on index (updated by isr) */
static void startMelodyNote(uint16_t idx)
{
    if (idx >= MELODY_LENGTH) {
        idx = 0;
    }

    gCurrentMelodyIndex = idx;
    uint32_t freq = gMelodyFreqs[idx];
```

```
    setToneFrequency(freq);

    /* compute note duration */
    uint32_t durMs = gMelodyDurMs[idx];
    gNoteSamplesRemaining = (durMs * gSampleRate) / 1000u;

    /* if duration is zero, click note for one sample*/
    if (gNoteSamplesRemaining == 0) {
        gNoteSamplesRemaining = 1;
    }
}

int main(void)
{
    SYSCFG_DL_init();
    /* Start the first note of the melody */
    startMelodyNote(0);
    /* Start timer to trigger DAC */
    DL_TimerG_startCounter(TIMER_0_INST);
    /* Enable DAC interrupt to update output */
    NVIC_EnableIRQ(DAC12_INT_IRQN);


    /* Calling WFI after calling DL_SYSCTL_enableSleepOnExit will result in
     * only ISR code to be executed. This is done to showcase the device's
     * low power consumption when sleeping.
     */
    DL_SYSCTL_enableSleepOnExit();
    while (1) {
        __WFI();
    }
}

void DAC12_IRQHandler(void)
{
    switch (DL_DAC12_getPendingInterrupt(DAC0)) {
        case DL_DAC12_IIDX_FIFO_1_2_EMPTY:
            {
                /* code for rest */
                if (gPhaseIncrement == 0) {
                    DL_DAC12_output12(DAC0, 2048);
                } else { /*code for tone, walk through phase table */
                    uint32_t tableIndex = (gPhaseAccumulator >> PHASE_FRAC_BITS) & (SINE_POINTS -
1);
                    DL_DAC12_output12(DAC0, gSineTable[tableIndex]);
                    gPhaseAccumulator += gPhaseIncrement;
                }

                /* state machine forward */
                if (gNoteSamplesRemaining > 0) {
                    gNoteSamplesRemaining--;
                }

                if (gNoteSamplesRemaining == 0) {
                    /* move to next note, wrap around */
                    uint16_t next = gCurrentMelodyIndex + 1;

                    /* causes song to loop*/
                    if (next >= MELODY_LENGTH) {
                        next = 0;
                    }
                    startMelodyNote(next);
                }
            }
            break;
        /* unused */
        case DL_DAC12_IIDX_FIFO_3_4_EMPTY:
        case DL_DAC12_IIDX_NO_INT:
        case DL_DAC12_IIDX_MODULE_READY:
        case DL_DAC12_IIDX_FIFO_FULL:
        case DL_DAC12_IIDX_FIFO_1_4_EMPTY:
        case DL_DAC12_IIDX_FIFO_EMPTY:
        case DL_DAC12_IIDX_FIFO_UNDERRUN:
        case DL_DAC12_IIDX_DMA_DONE:
            break;
    }
}
```

8 Additional Resources

- Texas Instruments, MSPM0 G-Series 80MHz Microcontrollers, technical reference manual.
- Texas Instruments, MSPM0G350x Mixed-Signal Microcontrollers with CAN-FD Interface, datasheet.

9 E2E

See TI's E2E™ support forums to view discussions and post new threads to get technical support for using MSPM0 devices in designs.

10 Trademarks

All trademarks are the property of their respective owners.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025