Subsystem Design

数模转换器到压电蜂鸣器

1 说明

该子系统演示了如何使用 MSPM0 的数模转换器 (DAC) 在压电式蜂鸣器上创建音调。此过程使用输出到压电执行器的预定义频率来创建调谐或叮当声，这在家用电器、工业机械或个人电子产品等应用中非常有用。

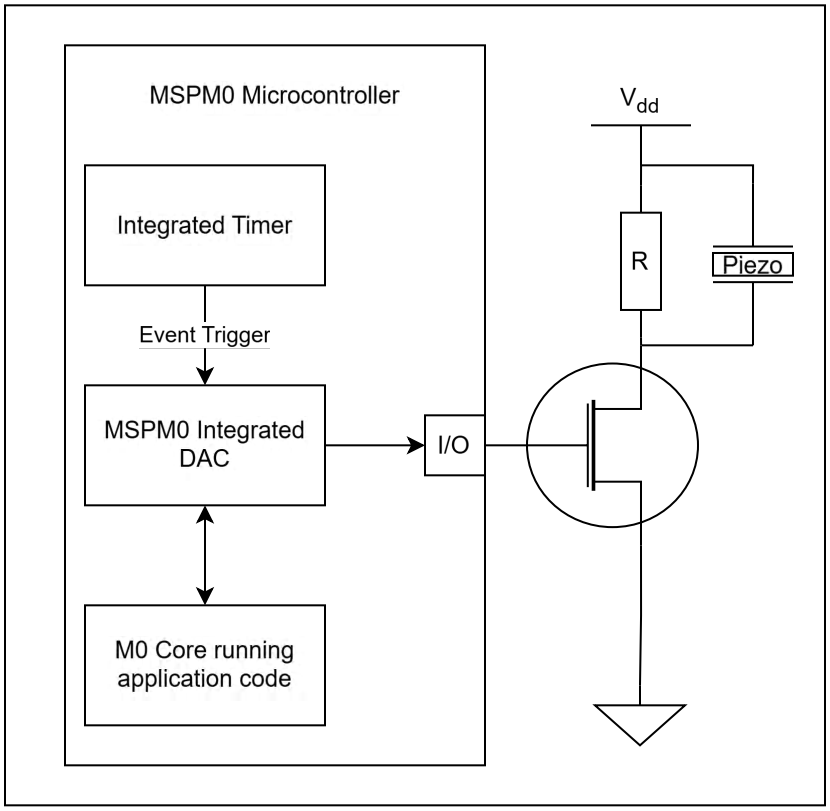

图 1-1. DAC 至蜂鸣器子系统方框图

2 所需外设

表 2-1. 所需外设

子块功能	外设使用	注释
音调发生器	DAC (DAC0)	12 位 DAC 从 64 点查找表输出正弦波样本，使用具有 24 个小数位的相位累加器生成音调以生成频率。
时序和采样速率控制	TimerG (TIMER_0_INST)	计时器根据采样率触发 DAC (示例代码中为 16.39，以适应 61us 周期) 。计时器事件触发 DAC FIFO 更新。
旋律排序	DAC ISR (FIFO_1_2_EMPTY)	状态机通过 ISR 前进，完成用户定义的旋律。
电源管理	SYSCTL	启用“退出时睡眠”以实现低功耗运行。器件仅在启动和 ISR 执行期间处于运行状态。

3 兼容器件

该子系统是使用 LP-MSPM0G3507 设计而成。所有符合所需外设部分所述功能的 MSPM0 器件都兼容，但会更改计时器、引脚选择和频率。

值得注意的是，根据所使用的器件，DAC_12 的 API 包装器可能会有所不同，因此需要对 ISR 进行细微调整，但功能不会改变。

4 设计步骤

1. 根据要实现的压电蜂鸣器，确定音频参数，包括采样率、波形类型、查找表分辨率和 DAC 分辨率。
2. 定义播放音符的频率（例如 NOTE_C4 = 262Hz），将它们组织成一个旋律，并定义每个音符的持续时间（以毫秒为单位）。
3. 计算计时器周期以生成所需的采样率触发事件。对于 10kHz 采样率，计时器必须以 20kHz 的频率触发。
4. 通过预计算扩展到板载 DAC 12 位范围 (0-4095) 的 256 个正弦值，将中点设置为 2048，与蜂鸣器进行正确的交流耦合，从而生成正弦波查找表。
5. 在 SysConfig 中，将计时器配置为事件发布者以触发 DAC12 模块，将 DAC 设置为在 FIFO 模式下运行（为半空事件启用中断），并将 DAC 输出路由到连接到蜂鸣器的相应 GPIO 引脚，如图 1-1 所示。
6. 编写应用程序代码来初始化正弦表，实施旋律状态机以跟踪要播放的当前注释和剩余注释，实现 DAC 中断处理程序以输出正弦值并推进旋律序列，最后启用退出睡眠模式以在采样之间实现低功耗运行。

5 设计注意事项

- **压电和音调频率选择：**必须根据连接到电路的压电蜂鸣器的属性对代码进行编辑。用于设计和配置所提供代码的压电蜂鸣器是 PS1230P02BT。采样率以及音调本身的频率必须根据任何压电的频率特性进行更改。
- **正弦表生成：**在示例代码中，使用基本的 64 条目正弦表在 DAC 上创建输出。但是，这会产生较低的输出质量，因此可以扩展到更高的分辨率，以获得更清晰的输出。建议正弦表是一个预定义的常量数组，或者在使用 MathACL 模块启动时生成一次。
- **采样率和计时器：**该子系统在代码中声明为 gSampleRate 的采样率必须等于计时器模块的周期。例如，如果所需的 gSampleRate 为 20kHz，则计时器周期必须为 0.05ms。提高采样率可以提供更好的音频质量和更平滑的正弦波再现。

6 软件流程图

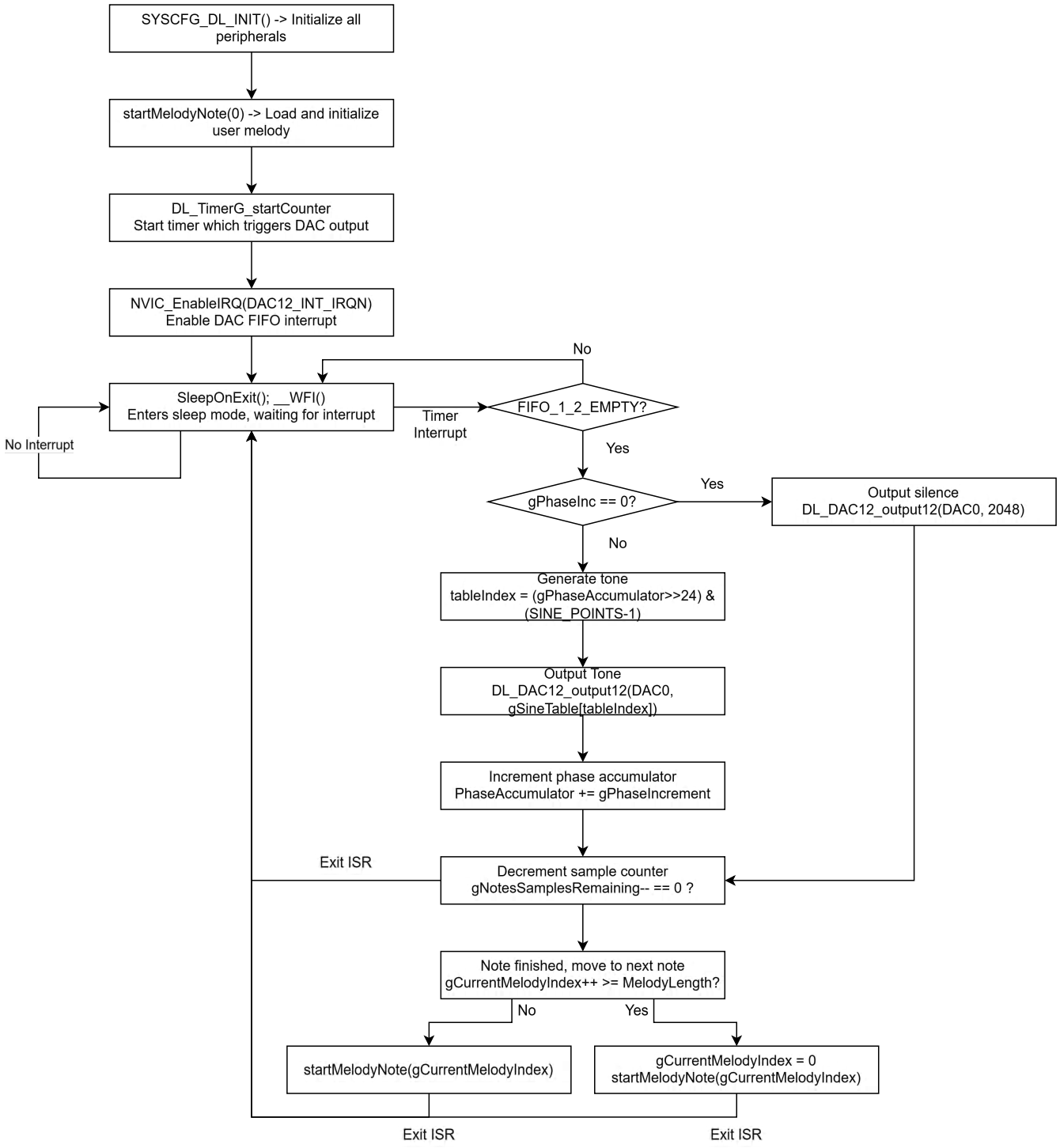

图 6-1. 软件流程图

7 应用代码

```
#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 其他资源

- 德州仪器 (TI)，MSPM0 G 系列 80MHz 微控制器技术参考手册。
- 德州仪器 (TI)，MSPM0G350x 具有 CAN-FD 接口的混合信号微控制器数据表。

9 E2E

请访问 TI 的 E2E™ 支持论坛来查看讨论并发布新主题，以获得在设计中使用 MSPM0 器件的技术支持。

10 商标

所有商标均为其各自所有者的财产。

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月