Application Note
ZHCAAE1 – 5. 2021

# DRV2625 触觉效果组合设计

*Mason Liu* *Sales and Marketing/East China*

**ABSTRACT**

随着技术的发展，越来越多的电子产品开始给用户提供反馈，包括听觉反馈，视觉反馈等等，而触觉反馈是一种新型的，不同于以上两种的反馈技术，可以解决在声音嘈杂以及光线受限的环境中，听觉反馈和视觉反馈无法及时反馈用户的问题。DRV2625 是德州仪器半导体公司(Texas Instruments)推出的一款驱动器，可以驱动转子马达(ERM)或者线性马达(LRA)。该芯片具有自动待机功能以及电池保护功能，内置有波形库以及可循环波形序列器，并且带有自动过驱和制动功能，可轻松的生成清晰优质的触觉效果，同时也简化了设计难度。

本文主要介绍了如何利用 DRV2625 设计出多种触觉效果，以及如何实现多种触觉效果的整合，并且给出了实际触觉效果设计的例子，使用户更加了解 DRV2625，并且能够实现触觉效果的设计。

**Contents**



**Figures**

# 1 如何利用 DRV2625 设计触觉效果

实际使用过程中，客户经常为了实验不同的触觉效果而大费脑筋，需要编写大量的代码，再进行一步步的调试，最终才能够得到一个满意的触觉效果。

而利用 TI 的 DRV2625 可以大大缩减编写代码以及调试的时间，DRV2625 中内置有第三方公司 Immersion 设计的波形库，通过调用波形库里的不同波形，就可以实现不同的触觉效果。DRV2625 在 ROM 中自带了 2 个波形库，第一个波形库(Lib A)适用于线性马达(LRA)，工作在闭环模式，第二个波形库(Lib B)适用于转子马达(ERM)，工作在开环模式。波形库的选择可以通过寄存器 0x0D 中的 bit 7 位：LIB_SEL 来实现。LIB_SEL=0，选择 Lib A；LIB_SEL=1，选择 Lib B。

DRV2625 驱动线性马达的驱动电压波形如图 1 所示。驱动电压波形可以分成三大部分：

1) 过驱电压，在启动阶段，DRV2625 给马达提供了一个大于额定电压的驱动电压，使马达能够快速启动，这一段对应的时间称为过驱时间。

2) 驱动保持电压，马达启动完成之后，电压也会随之下降，此时马达进入一个稳定的状态，这一段时间称为驱动时间。

3) 制动电压。最后当需要停止马达时，会通过给马达施加一个反向的驱动电压，使马达更快的克服惯性，使速度减为零，这一段时间称为制动时间。

改变上述的电压以及时间，都可以得到不同的触觉效果。

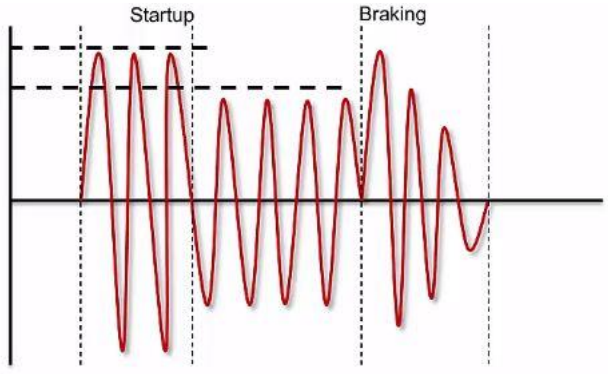


**Figure 1.　驱动电压波形**

以下图 2 中 Buzz 波形为例，加速度波形幅值的大小可以反映出震动波形的强，中等，微弱等震动强度。加速度波形的持续时间则是反映震动的时间长度：短促，中等，长等。这些不同的效果取决与不同的驱动电压和驱动时间。

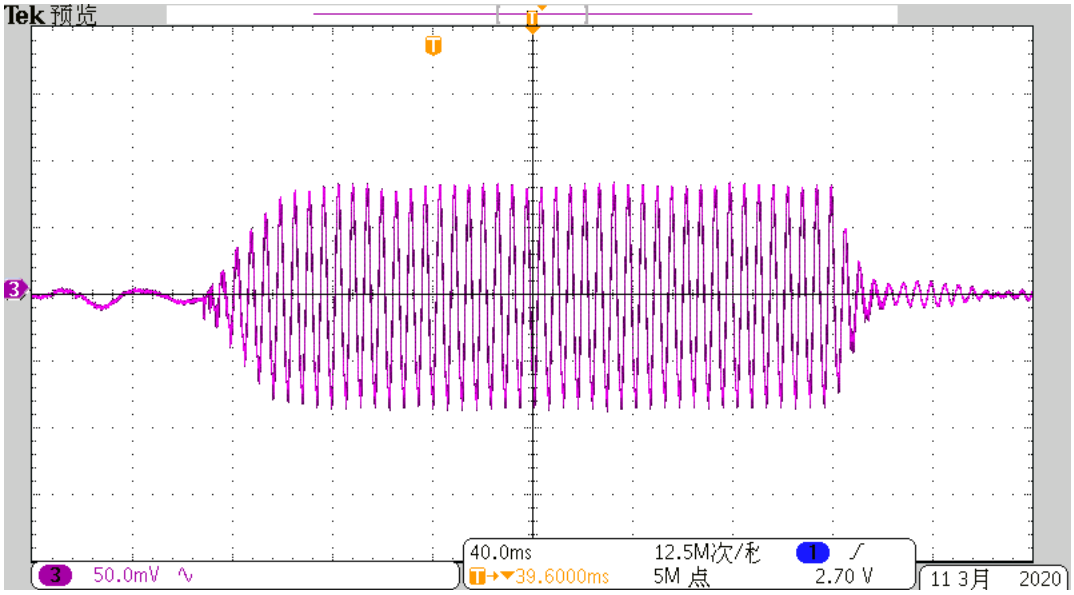


**Figure 2.　Buzz 加速度波形**

## 2　波形库介绍

在 DRV2625 中，波形库内部已经集成了 123 个不同震动时长，不同震动强度的波形，包括 Click, Bump, Buzz, Tick 等等。我们可以通过调用不同的波形，即可以满足不同的震动需求。

这里用 Sharp Click 波形以及 Sharp Tick 波形来举例分析，图 3 给出了波形库当中的 Sharp Click 的震动波形。Sharp Click - 100% 的震动强度会明显大于 Sharp Click - 60% 的震动强度。可以满足不同场景下不同的震度强度需求。

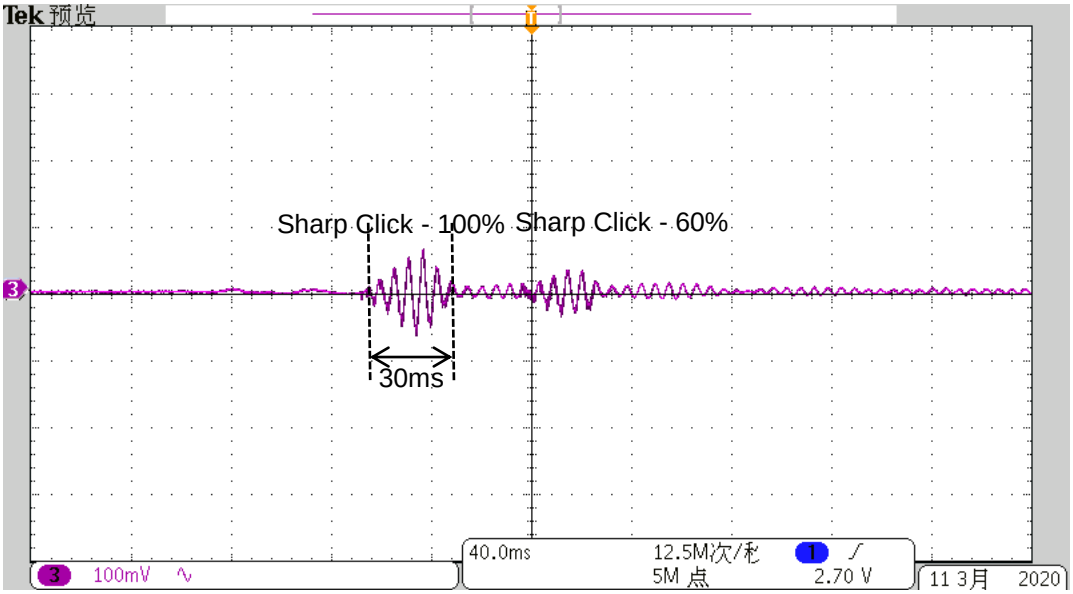

**Figure 3.　Sharp Click - 100%和 Sharp Click - 60%震动波形**

图 4 给出的是波形库当中的 Sharp Tick 的震动波形，Sharp Tick - 100%的震动时长为 20ms，Sharp Click - 100%的震动时长为 30ms，所以两者相比，我们可以发现，Sharp Tick 的震动波形更加短促一些。进而可以满足一些对于震动强度，震动时长有需求的一些应用场景。

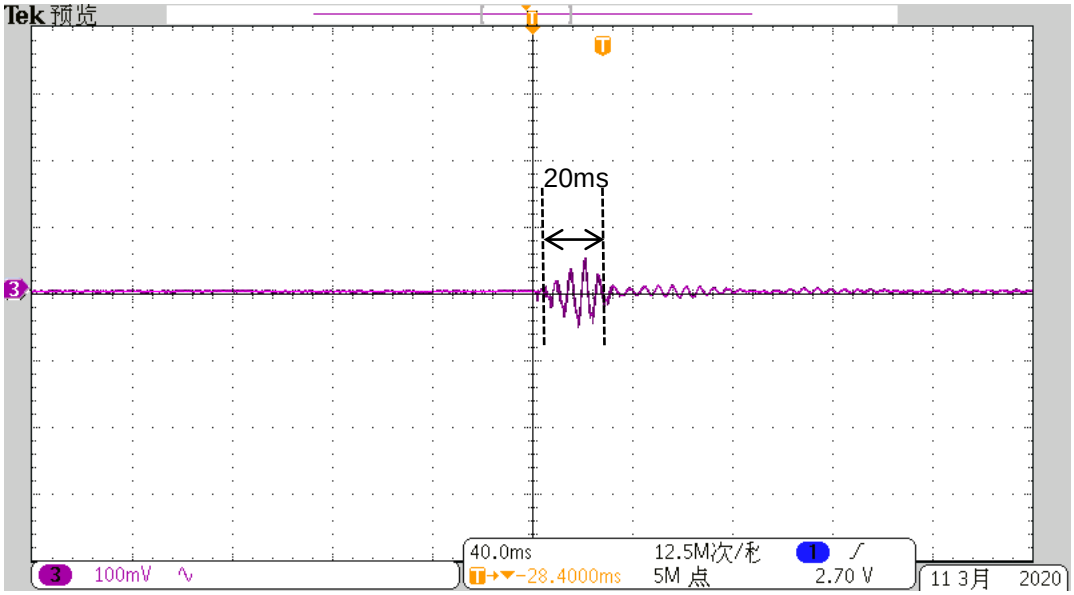


**Figure 4.　图 4 Sharp Tick - 100%震动波形**

# 3　多个触觉效果组合控制方法

DRV2625 内置了顺序控制寄存器(waveform sequencer registers)来控制各个不同波形播放的顺序以及中间间隔的时间，DRV2625 内部一共含有 8 个顺序控制寄存器。如下图 5 所示。其中 BIT 7：WAIT1~7 有两种含义：为 1 时表示这一个顺序控制寄存器用来做时间延迟，为 0 时表示这一个顺序控制寄存器用来触发不同的触觉效果，后面的 BIT0~BIT6 用来存储不同的震动波形。

| | | | |
|---|---|---|---|
| 0x0F | 0x01 | WAIT1 | WAV_FRM_SEQ1[6:0] |
| 0x10 | 0x00 | WAIT2 | WAV_FRM_SEQ2[6:0] |
| 0x11 | 0x00 | WAIT3 | WAV_FRM_SEQ3[6:0] |
| 0x12 | 0x00 | WAIT4 | WAV_FRM_SEQ4[6:0] |
| 0x13 | 0x00 | WAIT5 | WAV_FRM_SEQ5[6:0] |
| 0x14 | 0x00 | WAIT6 | WAV_FRM_SEQ6[6:0] |
| 0x15 | 0x00 | WAIT7 | WAV_FRM_SEQ7[6:0] |
| 0x16 | 0x00 | WAIT8 | WAV_FRM_SEQ8[6:0] |

**Figure 5.　顺序控制寄存器**

接下来就在此基础上详细分析如何实现多个触觉效果的组合控制。由于顺序控制寄存器存在长度限制，超出长度限制后可能会存在波形控制上的误差，本文主要讨论不同情况下该如何设计波形，并且推荐最合适设计方法。文中所有的波形均是在 DRV2625EVM-CT 上进行测试。

## 3.1　小于 8 个震动波形构成的触觉效果

第一种情况，最终的触觉效果由小于 8 个振动波型构成。举例说明，需要满足 Sharp Click - 100% + 20ms 延迟 + Sharp Tick - 100% + 20ms 延迟这样的触觉效果，经过分析，可以得到以下这样一张表格。在程序中如下图 6 所示：

表 1 范例 1

| 顺序控制寄存器地址 | 波形 |
|---|---|
| 0x0F | Sharp Click - 100% |
| 0x10 | 20ms 延迟 |
| 0x11 | Sharp Tick - 100% |
| 0x12 | 0x00 |
| 0x0C | 0x01 |

```
void DRV_Waveform_test4()
{
    Haptics_LoadSwitchSelect(Actuator_LRA);                                    //
    DRV_Enable();                                                              //
    DRV_Send_Actuator_Settings(Actuator_LRA);                                  //


    //Set up waveform sequencer
    StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqMainLoop, WaveSeqMainLoop_None, &hapticErr);

    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ1,SharpClick_100, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ2,Delay_20ms, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ3,SharpTick1_100, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ4,SharpClick_100, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ5,Delay_20ms, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ6,SharpTick1_100, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ7,0x00, &hapticErr);
//    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ8,Delay_150ms, &hapticErr);

    StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqLoop1to4, 0x00, &hapticErr);    // Repeat seq
    StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqLoop5to8, 0x00, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqMainLoop, 0x01, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, GOBIT, GO, &hapticErr);

    DRV_Poll();
}
```

**Figure 6.　范例 1 程序**

由于在 DRV2625 中以及集成好了每个波形， 所以我们在编写触觉效果组合程序时，只需要从库中调用对应的波形即可。这里我们在 0x0F 寄存器中调用 Sharp Click - 100%，在 0x10 寄存器中调用 20ms 延迟，在 0x11 寄存器中调用 Sharp Tick - 100%，最终按照此顺序便可以满足触觉效果的要求。接下来为了结束波形，我们首先需要在下一个顺序控制寄存器(0x12)中写入 0x00 来结束波形，最后需要将 GO BIT(0x0C)置位，等待下一次的触发。最终的触觉效果波形如下图 7 所示。CH3 为加速度波形，CH M 为 OUT+ 减 OUT-波形。

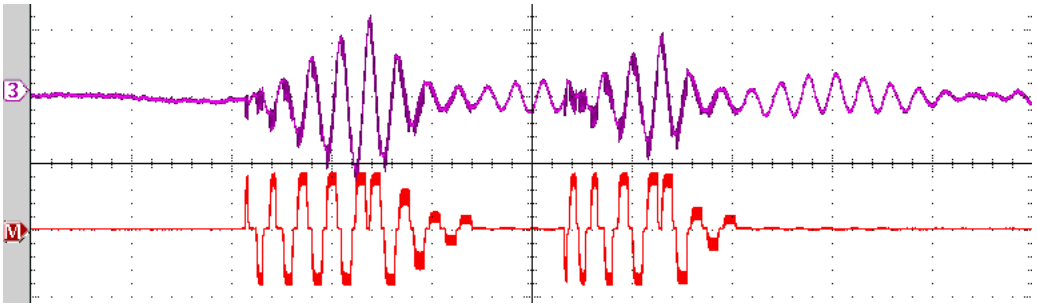


**Figure 7.　范例 1 震动波形**

## 3.2 含有重复波形的触觉效果

第二种情况，最终的触觉效果由小于 8 个振动波型构成，并且需要重复。同样接着上面的例子，需要满足 Sharp Click - 100% + 20ms 延迟 + Sharp Tick - 100%，并且重复 1 遍这样的触觉效果，经过分析，可以得到以下这样一张表格。

表 2 范例 2

| 顺序控制寄存器地址 | 波形 |
|---|---|
| 0x0F | Sharp Click - 100% |
| 0x10 | 20ms 延迟 |
| 0x11 | Sharp Tick - 100% |
| 0x12 | Sharp Click - 100% |
| 0x13 | 20ms 延迟 |
| 0x14 | Sharp Tick - 100% |
| 0x15 | 0x00 |
| 0x0C | 0x01 |

这张表格列出了所有的震动波型，可以发现比较繁琐，需要调用比较多的寄存器以及波形库里的波形，而在 DRV2625 中，已经配置好了一个用来多次循环的寄存器：WAV_SEQ_MAIN_LOOP，地址：0x19。通过这个寄存器，我们可以把上表简化为表 3，在程序中如下图 8 所示：

表 3 范例 2 化简

| 顺序控制寄存器地址 | 波形 |
|---|---|
| 0x0F | Sharp Click - 100% |
| 0x10 | 20ms 延迟 |
| 0x11 | Sharp Tick - 100% |
| 0x12 | 0x00 |
| 0x19 | 0x01 |
| 0x0C | 0x01 |

```
void DRV_Waveform_test3()
{
        Haptics_LoadSwitchSelect(Actuator_LRA);                                 //
        DRV_Enable();                                                           //
        DRV_Send_Actuator_Settings(Actuator_LRA);                               //


        //Set up waveform sequencer
        StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqMainLoop, WaveSeqMainLoop_None, &hapticErr);

        StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ1,SharpClick_100, &hapticErr);
        StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ2,Delay_20ms, &hapticErr);
        StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ3,SharpTick1_100, &hapticErr);
        StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ4,0x00, &hapticErr);
//        StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ5,StrongClick_100, &hapticErr);
//        StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ6,Delay_40ms, &hapticErr);
//        StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ7,StrongClick_100, &hapticErr);
//        StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ8,Delay_150ms, &hapticErr);

        StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqLoop1to4, 0x00, &hapticErr);    // Repeat sec
        StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqLoop5to8, 0x00, &hapticErr);
        StdI2C_P_TX_Single(DRV_ADDR, GOBIT, GO, &hapticErr);

        DRV_Poll();
}
```

**Figure 8. 范例 2 程序**

分析上表，可以看出通过配置 WAV_SEQ_MAIN_LOOP 为 0x01，减少了寄存器的使用量，加快了编写触觉效果程序的进程。最终的触觉效果如下图 9 所示。CH3 为加速度波形，CH M 为 OUT+ 减 OUT-波形。

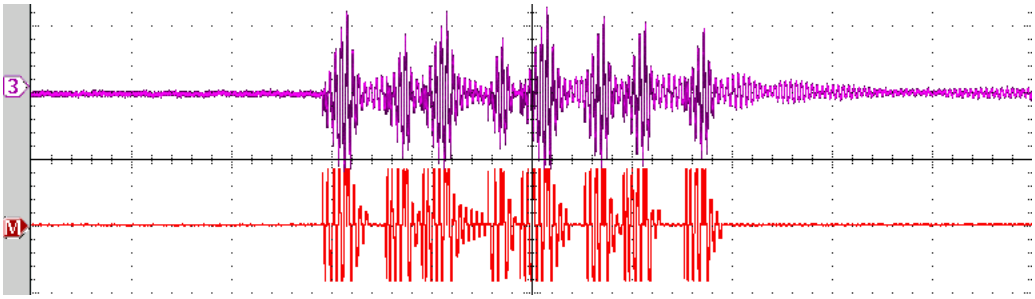

**Figure 9.　范例 2 震动波形**

## 3.3　大于 8 个震动波形构成的触觉效果

第三种情况，最终的触觉效果由大于 8 个震动波型构成。由于顺序控制寄存器只有 8 个，这里提供一种方法，能够满足上述情况。同样以上述例子为基础，需要满足 Sharp Click - 100% + 20ms 延迟 + Sharp Tick - 100% + 20ms 延迟+ Sharp Click - 100% + 20ms 延迟 + Sharp Tick - 80% + 20ms 延迟 + Sharp Click - 100% + 20ms 延迟，一共 10 个震动波形，我们可以通过连续调用两次顺序控制寄存器来解决顺序控制寄存器不够的问题。那么可以得到以下表格 4 和表格 5。

表 4 范例 3 第一部分

| 顺序控制寄存器地址 | 波形 |
|---|---|
| 0x0F | Sharp Click - 100% |
| 0x10 | 20ms 延迟 |
| 0x11 | Sharp Tick - 100% |
| 0x12 | 20ms 延迟 |
| 0x13 | Sharp Click - 100% |
| 0x14 | 20ms 延迟 |
| 0x15 | Sharp Tick - 100% |
| 0x16 | 20ms 延迟 |
| 0x0C | 0x01 |

表 5 范例 3 第二部分

| 顺序控制寄存器地址 | 波形 |
|---|---|
| 0x0F | Sharp Click - 100% |
| 0x10 | 20ms 延迟 |
| 0x11 | 0x00 |
| 0x0C | 0x01 |

在程序当中如下图 10 所示。其中 DRV_Waveform_test1()函数中是表 4 的触觉效果，DRV_Waveform_test2()函数中是表 5 的触觉效果。通过在不同的子函数中连续调用两次顺序控制寄存器，从而实现了由大于 8 个震动波型构成的触觉效果。但是此方法会存在一定的问题，两个子函数中间的时间会收到程序本身的影响，无法精确控制，所以主要推荐按照方式一和方式二来进行触觉效果的设计。

```
            case 0:
                DRV_Waveform_test1();
                DRV_Waveform_test2();
                break;

void DRV_Waveform_test1()
{
    Haptics_LoadSwitchSelect(Actuator_LRA);                                    //
    DRV_Enable();                                                              //
    DRV_Send_Actuator_Settings(Actuator_LRA);                                  //


    //Set up waveform sequencer
    StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqMainLoop, WaveSeqMainLoop_None, &hapticErr);

    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ1,SharpClick_100, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ2,Delay_20ms, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ3,SharpTick1_100, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ4,Delay_20ms, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ5,SharpClick_100, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ6,Delay_20ms, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ7,SharpTick1_100, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ8,Delay_20ms, &hapticErr);

    StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqLoop1to4, 0x00, &hapticErr);    // Repeat seq
    StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqLoop5to8, 0x00, &hapticErr);
//    StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqMainLoop, 0x01, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, GOBIT, GO, &hapticErr);

    DRV_Poll();
}

void DRV_Waveform_test2()
{
    Haptics_LoadSwitchSelect(Actuator_LRA);                                    //
    DRV_Enable();                                                              //
    DRV_Send_Actuator_Settings(Actuator_LRA);                                  //


    //Set up waveform sequencer
    StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqMainLoop, WaveSeqMainLoop_None, &hapticErr);

    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ1,SharpClick_100, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ2,Delay_20ms, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ3,0x00, &hapticErr);
//    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ4,Delay_20ms, &hapticErr);
//    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ5,SharpClick_60, &hapticErr);
//    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ6,Delay_20ms, &hapticErr);
//    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ7,SharpTick2_80, &hapticErr);
//    StdI2C_P_TX_Single(DRV_ADDR, Waveform_SEQ8,Delay_20ms, &hapticErr);

    StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqLoop1to4, 0x00, &hapticErr);    // Repeat seq
    StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqLoop5to8, 0x00, &hapticErr);
//    StdI2C_P_TX_Single(DRV_ADDR, WaveformSeqMainLoop, 0x01, &hapticErr);
    StdI2C_P_TX_Single(DRV_ADDR, GOBIT, GO, &hapticErr);

    DRV_Poll();
}
```

**Figure 10.　范例 3 程序**

最后下图 11 展示的就是方式三的触觉效果。CH3 为加速度波形，CH M 为 OUT+ 减 OUT-波形。图中标识出来的 time 即是上文中提到的无法精确控制的时间，可以看出我们程序中只有 20ms 的延时，但是实际波形中该延时达到了将近 80ms。

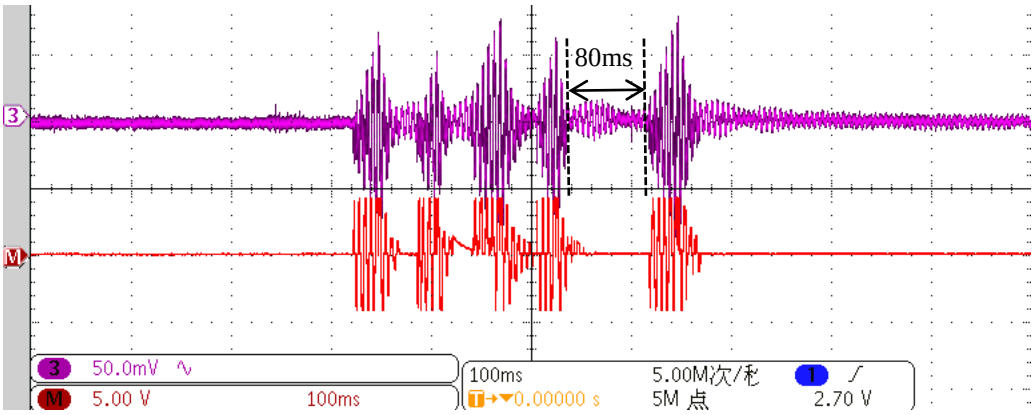


**Figure 11. 范例 3 震动波形**

综上所述，方法三中由于存在顺序控制器最大值的限制，所以延时时间会有所延长，建议采用方式一和二来进行波形的组合与调试，最终产生满足要求的触觉效果。

## 4 参考文献

1. *DRV2625 datasheet* (*SLOS879B*)
2. *DRV2625EVM User's Guide (SLOU515)*

# 重要声明和免责声明

TI 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、某特定用途方面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 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款 (https:www.ti.com.cn/zh-cn/legal/termsofsale.html) 或 ti.com.cn 上其他适用条款/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。

邮寄地址：上海市浦东新区世纪大道 1568 号中建大厦 32 楼，邮政编码：200122