2024 年春季学期单片机应用课程设计报告——

运行于 8051 单片机上的函数信号发生器

罗嘉宏*

2024 年 5 月 19 日

目录

*中国科学技术大学 化学与材料科学学院，安徽省 合肥市 金寨路 96 号，230026，e-mail: luojh@mail.ustc.edu.cn，学号: PB23030713，任课教师：梁晓雯

1 简介

1.1 背景

传统的模拟信号发生器主要基于模拟电路产生指定的信号，且这类信号主要是由无稳态振荡电路产生的方波信号演变而来（例如，三角波信号是通过方波信号对时间积分得到，或者先通过开关-电流源得到三角波信号再同时由开关状态得到方波信号，正弦波信号是由一个复杂的双极晶体管-电阻网络调理得到）。著名的 ICL8038 集成电路（由 Intersil 公司生产）就是这类模拟信号发生器的代表。图1 展示了该集成电路中的正弦波转换器设计。

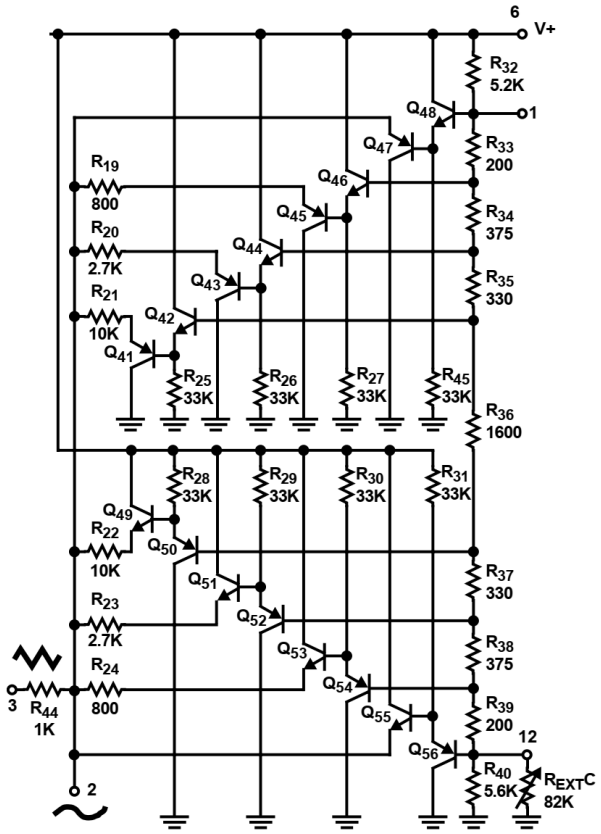

图 1: ICL8038 中的正弦波调理电路 (图片来源：ICL8038 数据手册，Harris Semiconductor)

然而，这类信号发生器具有很明显的缺点：

- 由于电路设计的原因，只能产生少数几种波形的信号；
- 由于电路依靠 R-C 或者 L-C 网络产生时间基准，时间基准的频率可变范围小，且可能带来大的误差；
- 正弦信号等信号由于是通过调理电路产生，与真正的正弦信号有差距（失真）；

- 只能通过可变电阻等方式调节信号参数，调节分辨率指标较差，且无法实现自动控制。

1.2 直接数字信号合成（DDS）的原理

使用数字电路产生模拟信号的方法称为直接数字合成（DDS），其主要原理为将波形数字化并将每个时间点上的数字化信号值传送到数字-模拟转换器（DAC），以逼近真实需要的波形。Nyquist 采样定理指出，在 DDS 输出的每一个周期内至少需要 2 个采样点才能重建出期望的输出波形。而实际上为了重建精确的期望波形，DAC 的输出刷新频率应该是期望频率的数十倍以上，具体取决于需要的重建精度。事实上，由于 DAC 输出的是类似于阶梯的波形，DAC 输出还需要低通滤波器才能得到理想的预期波形。

1.3 本设计介绍

为了解决上述的问题，本设计主要从几个方面入手：

- **任意信号的产生**：通过单片机计算出任意函数的值并周期性输出，完成任意波形的发生；
- **数值参数**：数值参数等通过数值键盘输入并通过 OLED 显示屏显示，实现数字化精确输入；
- **时间基准**：通过单片机的系统时钟中断产生（可以由晶体振荡器提供），提高了主时钟的精确性，可以通过分频产生大范围内的信号频率；
- **远程控制**：本设计除了可以使用键盘控制，还可以通过串行端口进行远程控制。

1.4 实现的硬件和软件

硬件 本设计的程序运行在 STC 公司生产的高速增强 8051 单片机上,型号为 `IAP15W4K58S4`。该单片机安装在皮赛电子生产的单片机开发板上。开发板上与本设计有关的硬件资源还有 LED 灯、128*64 像素 OLED 显示屏、DAC 转换器（`DAC081S101`，具有轨到轨输出的 8 位微功耗数模转换器）、4*4 矩阵键盘。

软件 本设计使用的开发工具是 Keil μVision，版本 V5.38.0.0。使用 `C51`，版本 9.60.7.0 编译器（工具链 `PK51 Prof. Developers Kit`，版本 V9.60.7.0）。

2 程序设计

2.1 程序结构

主程序 单片机开始运行后运行的是主程序，主程序负责控制用户界面、读取键盘事件并给出对应的响应。

定时器 0 中断程序（中断 1） 定时器 0 中断程序用于按照指定的模式将内存中准备好的信号数据传递到 DAC 中，得到对应的信号输出。

串行端口 1 中断程序（中断 4） 串行端口 1 中断程序用于响应串行端口接收完成 1 字节数据，以及发送完成 1 字节数据的事件。

三个程序的优先级关系 主程序从单片机上电复位开始就一直运行，但由于定时器 0 的中断频率非常高，主程序的执行会被不时打断。故主程序中没有安排时间关键性的任务，且定时器 0 的中断频率也需要精细调整防止其干扰用户的正常操作。定时器 0 中断负责按照精确的时间间隔（保证输出的准确性），将内存中准备好的数据传送到 DAC 上，故其优先级较高。串行端口 1 中断响应上位机发送的指令，由于要保证波特率和串行端口时序的准确性，此中断一旦触发，就会通过

```
if(serialReadAvailable)
{
    TR0 = 0;
    P7 &= 0xfe;
}
```

关闭定时器 1 中断，待远程命令处理完成之后再恢复输出。此时 DAC 将停止输出。考虑到串行端口的命令作用是调整输出波形参数，此时中止输出是合理的。

2.2 程序组成

本程序由多个文件组成，分别包含了不同的功能。下面列出了一些程序模块和它们的作用：

- `main.c`：主文件，包括了绝大部分的功能控制，例如：各级功能菜单、波形计算和生成、各种用户界面交互页（例如，输入电压、频率等的输入框）、自检、定时器中断（定时器 1）的中断处理函数。
- `buzzer.c`/`buzzer.h`：蜂鸣器控制函数。
- `DAC.c`/`DAC.h`：DAC 的控制函数。

- `keypad.c`/`keypad.h`：键盘扫描和处理功能。
- `oled.c`/`oled.h`：OLED 显示屏的控制函数。
- `uart.c`/`uart.h`：串行端口通信控制的函数、串行端口中断处理函数。
- `ustclogo.h`：中科大校名和 logo（在开机时显示）的存储。

2.3 程序结构

图 2 展示了各个模块之间的关系。

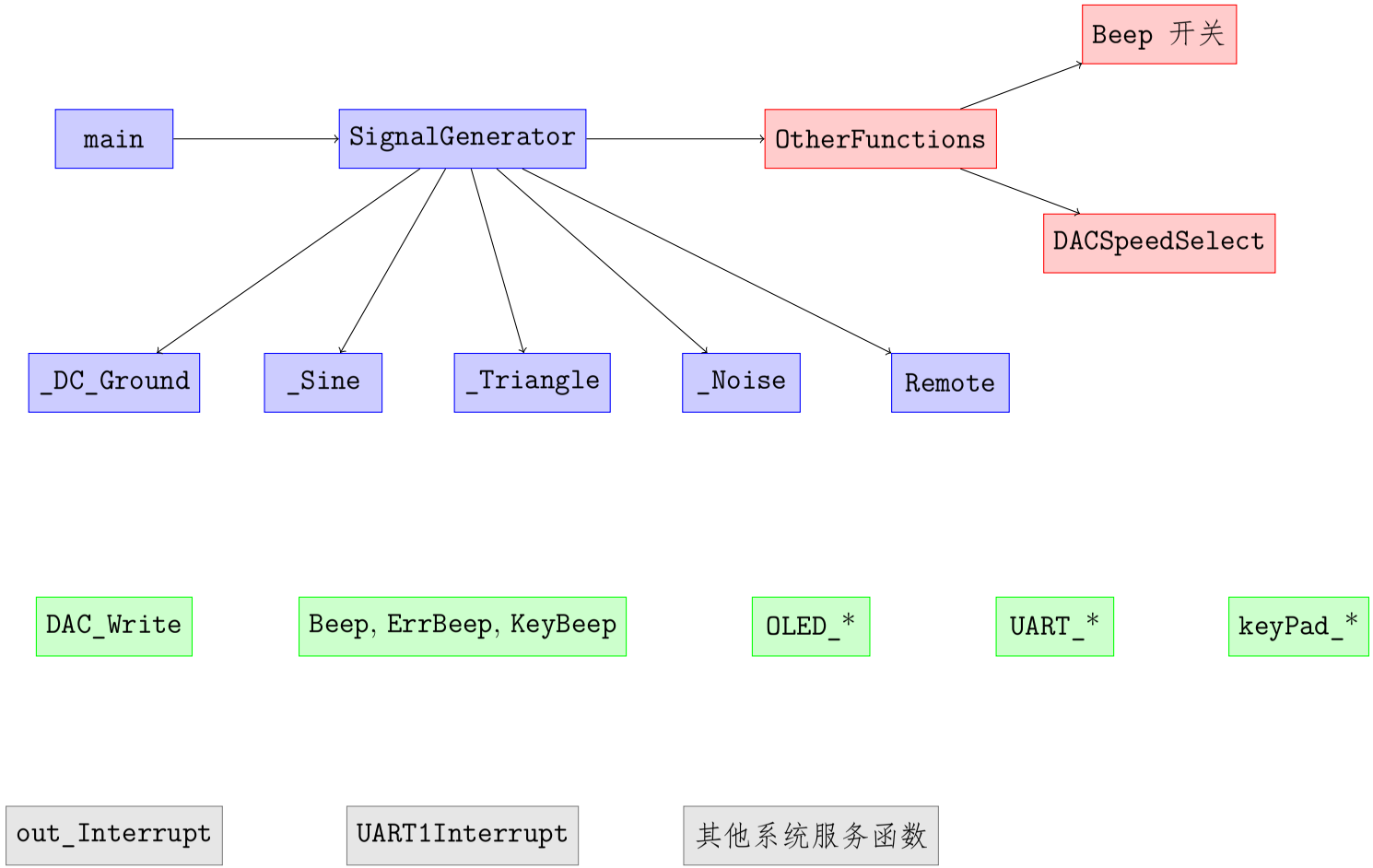
Beep 开关
main
SignalGenerator
OtherFunctions
DACSpeedSelect
_DC_Ground
_Sine
_Triangle
_Noise
Remote
DAC_Write
Beep, ErrBeep, KeyBeep
OLED_*
UART_*
keyPad_*
out_Interrupt
UART1Interrupt
其他系统服务函数

图 2: 程序结构的示意图

3 代码中的重点解析

3.1 字体反白实现

在菜单中选择项目时，以及显示标题条时，往往需要使用白底黑字的显示，这时候就需要进行一些技巧性的处理。注意到我们的字模数据是 8×16 大小的，且行优先取模。也就是说，字体是一行一行存储的，且每行刚好是一个字节。这样，由于这个字节中的“0”和“1”是表示点亮和熄灭对应像素点的两种状态，我们直接对每个字节位取反就可以实现反白显示。代码片段如下：

```
1  void OLED_ShowChar(u8 x, u8 y, u8 chr) {
2          // ...
3
4          OLED_Set_Pos(x, y);
5          for (i = 0; i < 8; i++)
6                  OLED_WR_Byte(F8X16[c * 16 + i], OLED_DATA);
7
8          OLED_Set_Pos(x, y + 1);
9          for (i = 0; i < 8; i++)
10                 OLED_WR_Byte(F8X16[c * 16 + i + 8], OLED_DATA);
11 }
12
13 // With inverted color
14 void OLED_ShowCharInverted(u8 x, u8 y, u8 chr) {
15         // ...
16
17         OLED_Set_Pos(x, y);
18         for (i = 0; i < 8; i++)
19                 OLED_WR_Byte(~F8X16[c * 16 + i], OLED_DATA);
20
21         OLED_Set_Pos(x, y + 1);
22         for (i = 0; i < 8; i++)
23                 OLED_WR_Byte(~F8X16[c * 16 + i + 8], OLED_DATA);
24 }
```

代码中，`F8X16[]` 即为字模数据，注意 `OLED_ShowCharInverted` 和 `OLED_ShowChar` 在 `OLED_WR_Byte` 中参数的不同。这里还有一个更好的方法，即直接用对应的字模数据对

0x00 或 0xff 取位异或。这是因为：

$$X' = X \oplus A = \begin{cases} X & , A = 0 \\ \bar{X} & , A = 1 \end{cases}$$

3.2 两个中断的处理

串口中断和输出波形刷新的中断会互相干扰。同样，输出中断对主程序对用户界面的刷新、处理也会有干扰。所以，我们采用了动态开启、关闭中断的策略。由于需要保证串口通信不被打断，故串口接收到第一个字节开始，就需要关闭输出中断。考虑到串口命令是用来改变输出参数的，所以这时中断输出也是可取的。这段代码如下：

```
1  uchar uartReadByteWithCancel() // 0--127 for byte, 0xf0 -- 0xff ↩
       for key cancel
2  {
3          uchar key, mod = 64;
4          while(1)
5          {
6                  if(serialReadAvailable == 1)
7                  {
8                          serialReadAvailable = 0;
9                          P7 &= 0xfe;
10                         TR0 = 0; // 关闭输出中断
11                         return SBUF;
12                 }
13                 if(mod == 64)
14                 {
15                         mod = 64;
16                         key = keypadReadKey();
17                         if(key != 0xff)
18                         {
19                                 KeyBeep();
20                                 return 0xf0 + key;
21                         }
22                 }
23         }
24 }
```

同时，还可以见到，我们还设计了在无限循环下，按键时终止串口读取的功能。这是为了可以按下 Esc 键退出远程控制模式。

为了保证频繁的输出中断不会干扰用户界面，我们对输出中断的频率进行了仔细的调节，最后确定了 10kHz 为最高的频率。这个频率下，用户界面基本不受干扰。

3.3 波形产生

波形信号的生成 波形信号通过特定的函数生成并写入到公用的存储区内，例如下面的代码用于生成正弦信号：

```
1  void FillSineBank(uchar uBias, uchar uAmp)
2  {
3          uchar x = 0;
4
5          OLED_Clear();
6          OLED_ShowStringRaw(0, 0, "Refresh␣Wavebank", 1);
7          OLED_ShowStringRaw(0, 2, "X=␣", 0);
8          OLED_ShowStringRaw(0, 4, "U=␣", 0);
9          OLED_ShowStringRaw(0, 6, "Type:␣Sine", 0);
10
11         for(x = 0; x < BANKSIZE; ++x)
12         {
13                 prepBank[x] = roundM((float)uBias + (float)uAmp * ↩
                       __sin((float)x / BANKSIZE * 2 * 3.1415926));
14                 OLED_ShowNumRaw(24, 2, x, 5, 16, 0);
15                 OLED_ShowNumRaw(24, 4, prepBank[x], 5, 16, 0);
16         }
17         Beep();
18 }
```

可见，prepBank 中的信号与频率无关。频率的调控依赖于以怎样的速度扫过并输出 prepBank 里面的内容。这样就避免了每次改频率都需要重新生成波形（这个过程比较慢）。而改变电压时却会需要每次重新生成波形，这是因为我们需要避免在波形发生时临时进行电压的比例计算（单片机上除法速度很慢）。

波形输出 波形输出（即刷新 DAC 数据）是在定时器中断中实现的，代码如下：

```
1  void out_Interrupt (void) interrupt 1
2  {
3          P7 ^= 0x04;
4
5          if(genNoise) // 噪声模式
```

```
6       {
7               ++waveI;
8               if(waveI >= wavePeriod)
9               {
10                      P7 ^= 0x02; // 指示灯
11                      DAC_Write(rand() % prepBank[0] + prepBank[1]);↩
                            // 写入DAC
12                      waveI = 0;
13              }
14      }
15      else // 读取prepBank的模式
16      {
17              DAC_Write(prepBank[(++waveI) * BANKSIZE / wavePeriod↩
                    ]); // 写入DAC
18
19              if(waveI >= wavePeriod - 1)
20              {
21                      // 一个周期结束
22                      waveI = 0;
23                      P7 ^= 0x02; // 指示灯
24              }
25      }
26
27      if(serialReadAvailable) // 有串口数据?
28      {
29              TR0 = 0;
30              P7 &= 0xfe; // 指示灯
31      }
32  }
```

可见，代码分成了两种情况处理：噪声模式和读取 prepBank 的模式。此外还处理了串口数据：如果有串口数据下发则停止输出。

4 工作展示

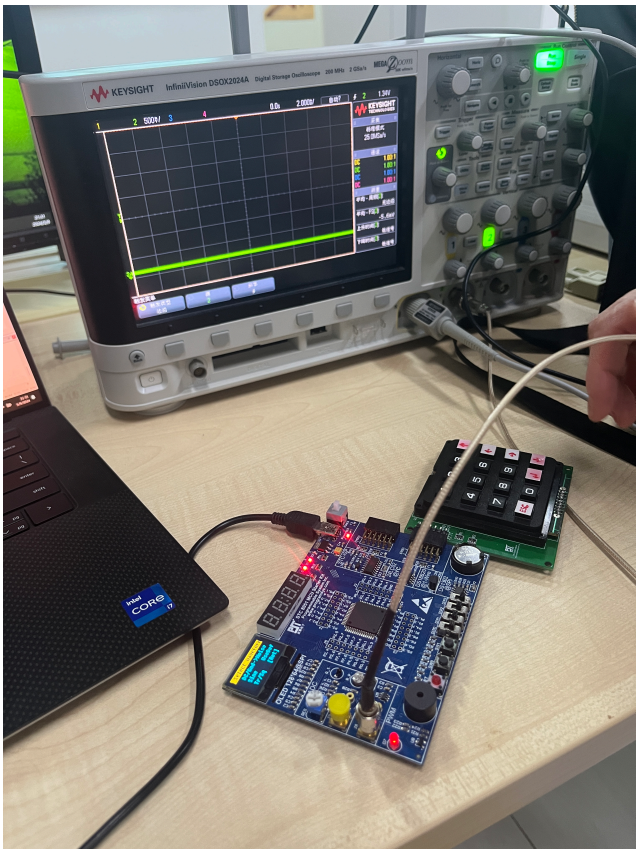

图 3: 装置整体

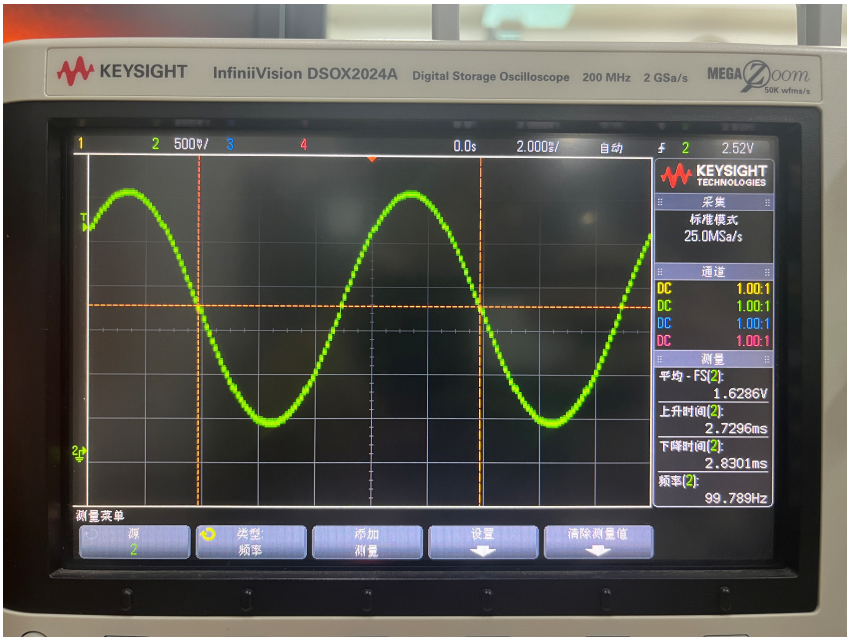

图 4: 100Hz 正弦波

图 5: 200Hz 正弦波

图 6: 三角波

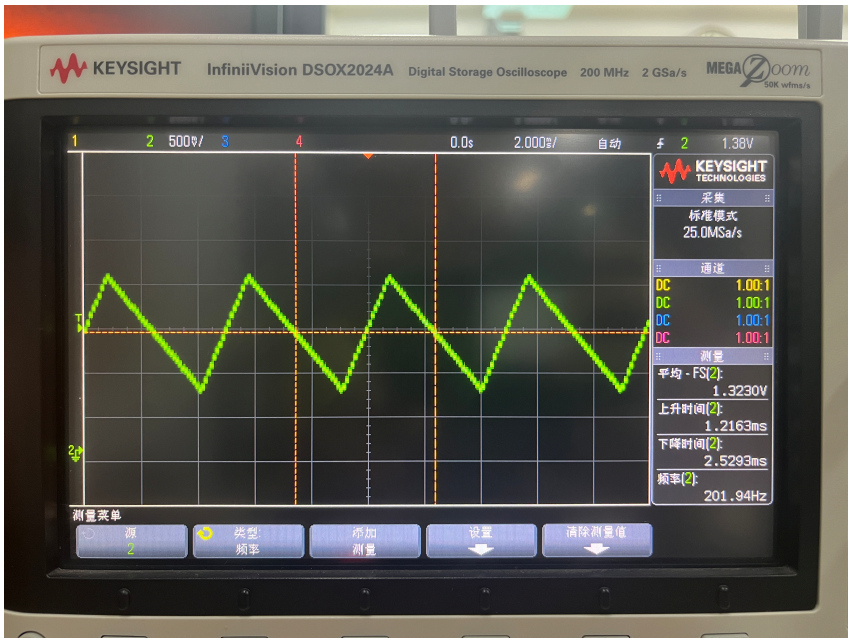

图 7: 三角波（另一个参数）

图 8: 方波

图 9: 噪声

图 10: 用户定义波形

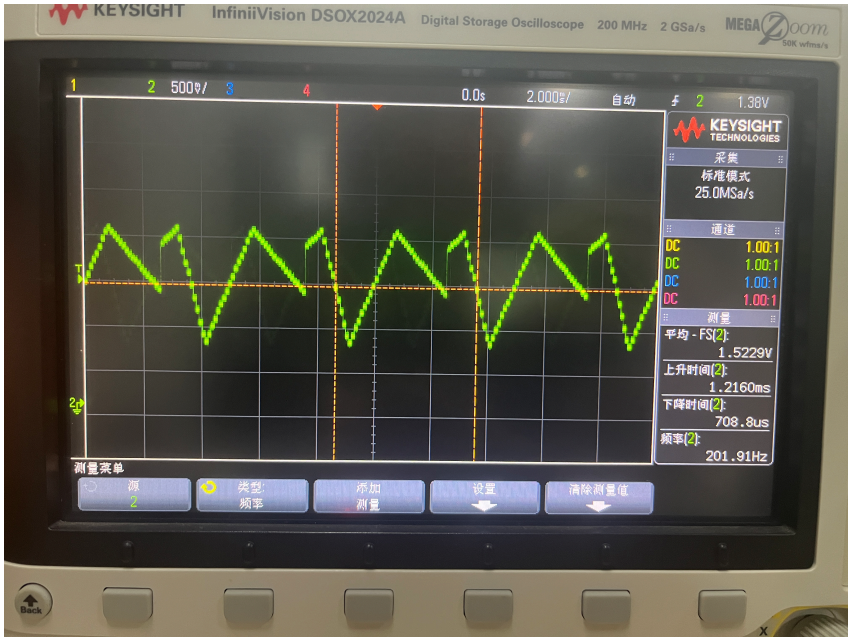

图 11: 波形参数变化时刷新过程