基于 80C51 单片机的波形发生器设计与仿真

徐　旭

(通信工程)

[摘要]本文使用 80C51 单片机作为主控芯片，通过按键控制跳转不同的中断子程序分别产生三角形波、方波和正弦波信号。选择数模转换芯片 DAC0832 将单片机产生的 8 位数字信号转换成模拟信号输出稳定的电流，并通过数模转换芯片输出端口（IOUT1、IOUT2）处连接的运算放大器将该模拟电流信号转换为电压波形。最后通过外接的示波器显示相应的波形，仿真结果显示，输出的三种波形稳定清晰。

[关键词]单片机课程设计；波形发生器设计；Protues7.8 **仿真**

一、绪论

1.1. 任务分析

本次设计是基于 80C51 单片机的波形发生器电路，通过单片机与总线、译码器、数模转换器 DAC0832 以及三只按键之间的配合，实现跳转不同的中断服务子程序分别产生方波、三角形波和正弦波的效果。在本设计中，80C51 单片机的主频为 12MHz，P1 口连接按键，P0 口连接数模转换芯片和译码芯片输出波形。

软件代码使用单片机汇编语言编写，并通过 Keil μVision4 软件编译产生十六进制执行文件（.hex）；硬件电路在 Proteus7.8 中搭建，配合代码中各芯片地址进行电路布线，在引入编译产生的可执行文件后对电路仿真。

1.2. 总体设计思路

在本设计中 80C51 单片机通过译码电路对数模转换芯片 DAC0832 进行片选，并通过 P0 口进行通信，设计思路框图如图 1 所示。查阅资料后，笔者决定采用外部不同按键的接通情况作为中断信号，促使单片机跳转执行不同的波形发生中断子程序，产生的数字信号通过 DAC0832 芯片转换成模拟信号最后输出。

根据课程教材以及网上参考资料，三角波的产生可以分为两个过程，首先寄存器从零开始加 1 产生上升边，寄存器溢出后开始递减产生下降边，减至零后循环该过程即可产生连续的三角形波；方波的产生较为简单，首先规定方波高电平对应的数值，调用延时子程序，接着定义方波低电平对应数值，调用延时子程序，重复该过程即可获得连续的方波信号；笔者采用查表法产生正弦波信号，即给出正弦值表，在程序中遍历这张数值表并输出，循环该过程即可。

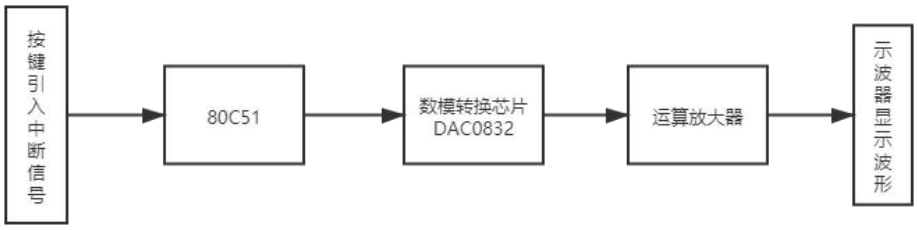


图 1 总体设计框图

二、硬件设计分析

图 2 为笔者在 Proteus7.8 软件中搭建的电路，使用到的元件包括一片 80C51 芯片、1 片 74LS373 芯片、两片 DAC0832 芯片、一个与门、两个运算放大器、三只按键和一个示波器。电路中三只按键由上至下触发后依次产生三角波、方波和正弦波，且标号为 U2 的数模转换芯片用以产生三角波模拟信号，标号为 U5 的数模芯片用以产生方波和正弦波模拟信号。

根据上述思路，本设计中需要使用 INT0 外部中断，而 INT0 中断地址为 0003H，设置 INT0 外部中断触发方式为边沿触发，打开单片机中断允许总开关，并且需要允许外部中断 INT0。

为方便电路布线，笔者设置通过 74LS373 的 Q1 引脚片选 U2 数模芯片，通过 Q0 引脚片选 U5 数模芯片，两芯片的地址如下表所示。

芯片标号	地址
U2	0FDH
U5	0FEH

表 1 数模芯片地址

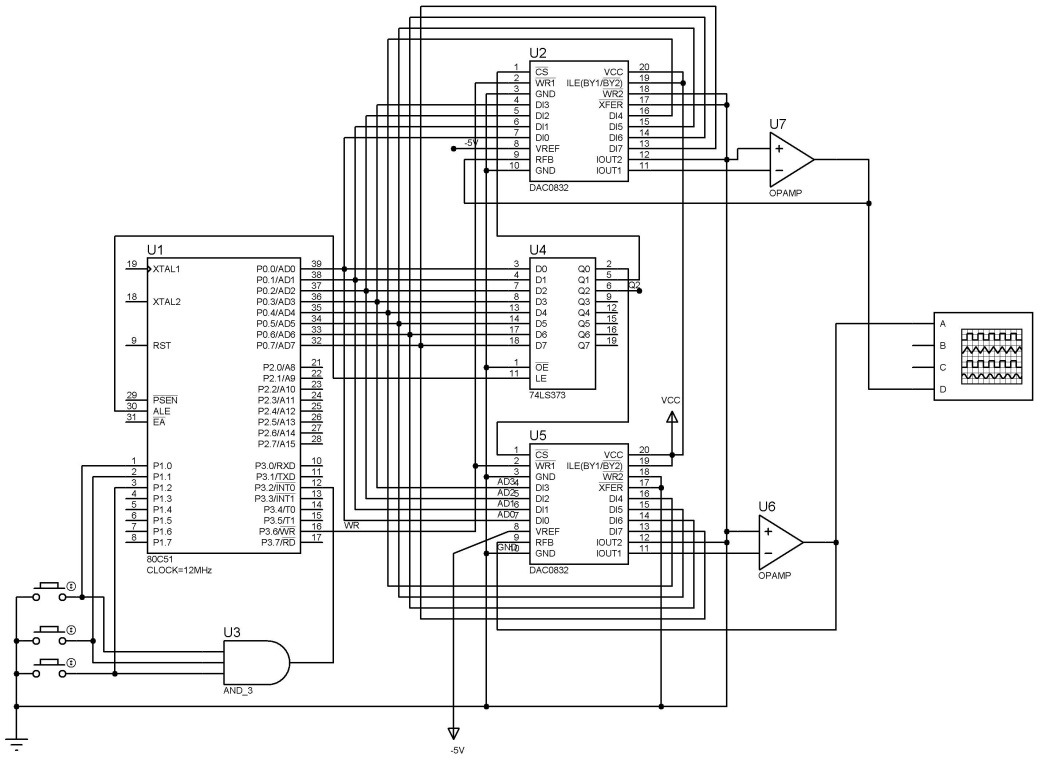


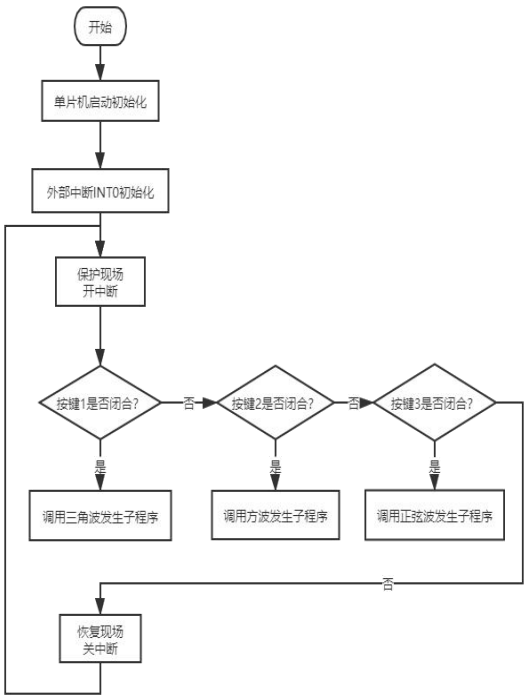
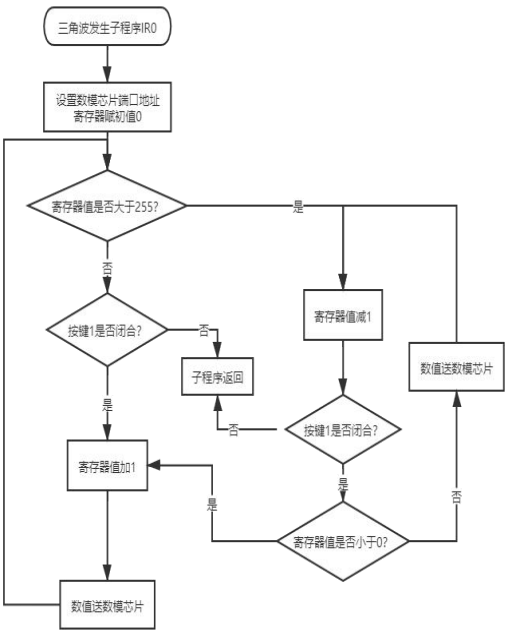
图 2 硬件电路图

三、软件设计分析

3.1. 流程图设计

根据前面硬件电路的设计以及总体思路要求，主程序和三种波形发生子程序的流程图如表 2 所示。由于硬件采用按压-弹起式按钮，每次最多只能引入一个中断子程序，不会产生多个中断子程序同时被调用的情况。

本次设计中，笔者使用一个简单的递减循环作为方波信号高低电平的延时子程序，为了简化设计便于理解，循环次数共计为 25×10=250 次。

程序名称	流程图
主程序	
三角波发生子程序	

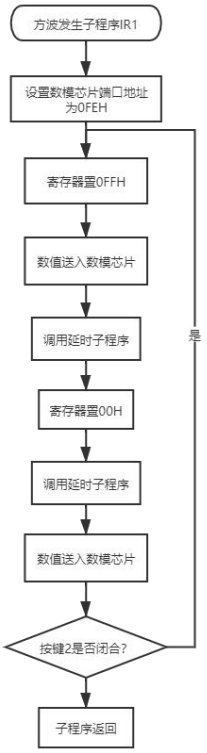
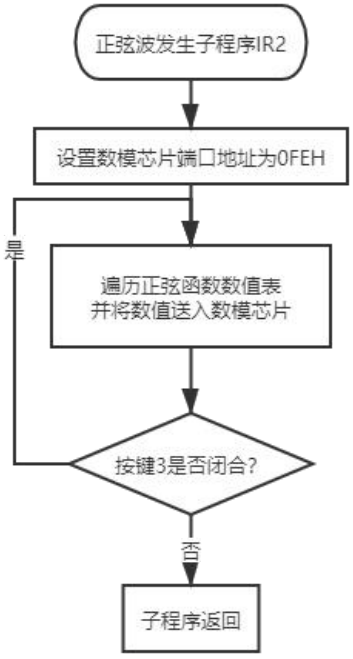
方波发生子程序	
正弦波发生子程序	

表 2 程序流程图

3.2. 程序代码

```
1.  ORG 0000H
2.  LJMP MAIN
```

```
3.      ORG 0003H
4.      LJMP INT00
5.      ORG 0040H
6.   MAIN:
7.      MOV SP,60H
8.      SETB IT0
9.      SETB EA
10.     SETB EX0
11.  INT00:
12.     CLR EA
13.     PUSH PSW
14.     PUSH Acc
15.     SETB EA
16.     JNB P1.0,IR0
17.     JNB P1.1,IR1
18.     JNB P1.2,IR2
19.  INTIR:
20.     CLR EA
21.     POP Acc
22.     POP PSW
23.     SETB EA
24.     RETI
25.
26.  IR0:
27.     MOV R0,#0FDH
28.     MOV A,#00H
29.  UP0:
30.     MOVX @R0,A
31.     JB P1.0,INTIR
32.     INC A
33.     JNZ UP0
34.  DOWN0:
35.     DEC A
36.     MOVX @R0,A
37.     JB P1.0,INTIR
38.     JNZ DOWN0
39.     SJMP UP0
40.     LJMP INTIR
41.
42.  IR1:
43.     MOV R0,#0FEH
44.  UP1:
45.     MOV A,#0FFH
46.     MOVX @R0,A
```

```
    LCALL DELAY
    JB P1.1,INTIR
DOWN1:
    MOV A,#00H
    MOVX @R0,A
    LCALL DELAY
    JB P1.1,INTIR
    SJMP UP1
    LJMP INTIR

IR2:
    MOV R0,#0FEH
    MOV R1,#00H
    MOV A,#00H
    MOV DPTR,#00H

SEARCH:
    MOV A,R1
    LCALL VALUE_SEARCH
    MOVX @R0,A
    JB P1.2,INTIR
    INC R1
    SJMP SEARCH
    LJMP INTIR


VALUE_SEARCH:
    MOV DPTR,#TABLE
    MOVC A,@A+DPTR
    RET

DELAY:
    MOV R7,#25
D1:
    MOV R6,#10
D2:
    DJNZ R6,D2
    DJNZ R7,D1
    RET

TABLE:
    DB 080h,083h,086h,089h,08ch,090h,093h,096h,099h,09ch,09fh,0a2h,0a5h,0a8h,0abh,0aeh
    DB 0b1h,0b3h,0b6h,0b9h,0bch,0bfh,0c1h,0c4h,0c7h,0c9h,0cch,0ceh,0d1h,0d3h,0d5h,0d8h
    DB 0dah,0dch,0deh,0e0h,0e2h,0e4h,0e6h,0e8h,0eah,0ebh,0edh,0efh,0f0h,0f1h,0f3h,0f4h
```

```
91.  DB 0f5h,0f6h,0f8h,0f9h,0fah,0fah,0fbh,0fch,0fdh,0fdh,0feh,0feh,0feh,0ffh,0ffh,0ffh
92.  DB 0ffh,0ffh,0ffh,0ffh,0feh,0feh,0feh,0fdh,0fdh,0fch,0fbh,0fah,0fah,0f9h,0f8h,0f6h
93.  DB 0f5h,0f4h,0f3h,0f1h,0f0h,0efh,0edh,0ebh,0eah,0e8h,0e6h,0e4h,0e2h,0e0h,0deh,0dch
94.  DB 0dah,0d8h,0d5h,0d3h,0d1h,0ceh,0cch,0c9h,0c7h,0c4h,0c1h,0bfh,0bch,0b9h,0b6h,0b3h
95.  DB 0b1h,0aeh,0abh,0a8h,0a5h,0a2h,09fh,09ch,099h,096h,093h,090h,08ch,089h,086h,083h
96.  DB 080h,07dh,07ah,077h,074h,070h,06dh,06ah,067h,064h,061h,05eh,05bh,058h,055h,052h
97.  DB 04fh,04dh,04ah,047h,044h,041h,03fh,03ch,039h,037h,034h,032h,02fh,02dh,02bh,028h
98.  DB 026h,024h,022h,020h,01eh,01ch,01ah,018h,016h,015h,013h,011h,010h,00fh,00dh,00ch
99.  DB 00bh,00ah,008h,007h,006h,006h,005h,004h,003h,003h,002h,002h,002h,001h,001h,000h
100.   DB 000h,000h,001h,001h,002h,002h,002h,003h,003h,004h,005h,006h,006h,007h,008h,00ah
101.   DB 00bh,00ch,00dh,00fh,010h,011h,013h,015h,016h,018h,01ah,01ch,01eh,020h,022h,024h
102.   DB 026h,028h,02bh,02dh,02fh,032h,034h,037h,039h,03ch,03fh,041h,044h,047h,04ah,04dh
103.   DB 04fh,052h,055h,058h,05bh,05eh,061h,064h,067h,06ah,06dh,070h,074h,077h,07ah,07dh
104.
105.   END
```

四、仿真结果

汇编程序在 Keil μVision4 软件中成功编译，将 HEX 文件写入 Proteus7.8 软件的 80C51 单片机中仿真后发现，三种输出波形如图 5 所示。

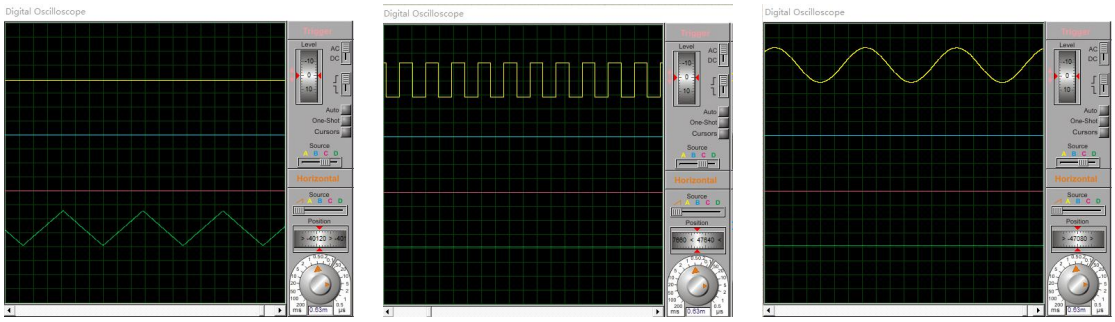


图 3 三种输出波形

五、分析体会

基于单片机的波形发生电路使用 80C51 单片机驱动数模转换器 DAC0832，将程序产生的波形数字信号转换为模拟电流信号，通过运算放大器后输出对应的电压波形并利用示波器进行展示。

本次设计中单片机使用的汇编语言与传统 8086CPU 使用到的汇编代码格

式、指令等不尽相同，所以在软件编写调试时查阅了大量资料和参考文献。但是汇编语言十分底层，有助于对单片机寄存器组以及中断子程序调用过程的理解，这对我们理解教材知识十分有益。

课程设计是一个发现错误，并通过软硬件联动调试程序解决错误的过程，我在调试设计过程中遇到了一些问题：受 8086 汇编格式影响，笔者将正弦函数的数值表放在了主程序的起始位置，但是发现示波器中显示的正弦中存在噪声，注释掉所有正弦函数值仅留存标号仍有噪声存留，多次调试后将数值表放在了整个程序的末尾，噪声才彻底消失。

笔者猜测，可能是数值表的存放位置对一些寄存器的值产生影响，进而使整个正弦函数发生子程序的输出波形因存在杂波而失真。单片机部分的内容实用性很强，值得课余投入更多时间进行更加深入的研究。