

[设为首页](#) [收藏本站](#)

用QQ帐号登录

帐号 用户名/Email

☐ 自动登录[找回密码](#)

只需一步，快速开始

密码

[注册](#)[论坛](#)[语法手册](#)[淘宝杂货铺](#)[官方微博](#)[官方QQ群](#)[关于我们](#)[快捷导航](#)

请输入搜索内容

帖子



热搜: mpu6050 蓝牙 串口 PID 12864 红外 小车 MPU6050 1602 GPS 舵机 最小系统 android

[论坛](#) > [热门MCU](#) > [Arduino](#) > 有限状态机在单片机和 Arduino 编程中的应用 ...[发帖](#)[返回列表](#)

1

2

1 / 2 页

[下一页](#)

查看: 27702 | 回复: 22

有限状态机在单片机和 Arduino 编程中的应用 [复制链接]



Blanboom

发表于 2015-2-8 22:59:58 | 只看该作者 | 只看大图



1# 电梯直达

本帖最后由 Blanboom 于 2015-3-4 20:18 编辑

在单片机编程中，如果在不使用操作系统的情况下同时执行多个任务，可能会遇到下面这些情况：

- 一个任务的执行时间过长，导致其他任务无法及时执行
- 在一些任务中大量使用 delay() 等函数进行软件延时，这些延时函数占用过多时间，影响其他任务的执行
- 一些复杂任务的程序逻辑不清晰，不便于以后对程序进行维护，或添加新功能

本文介绍的有限状态机，可以做到将一个耗时较多的复杂任务分解为多个简单任务，同时使代码逻辑更加清晰，从而解决上述问题。

(本文原载于我的博客，转载时请保留此链接：<http://blanboom.org/finite-state-machine-in-microcontrollers.html>)

目录：

- 1. 什么是有限状态机
- 2. 有限状态机的作用
 - 2.1 分解耗时过长的任务
 - 2.2 避免软件延时对 CPU 资源造成浪费
 - 2.3 使程序逻辑更加清晰
- 3. 有限状态机的实现
 - 3.1 通过 switch - case 语句实现
 - 3.2 通过 Arduino 库实现
 - 3.3 其他方式
- 4. 示例一：按键去抖动程序的优化
 - 4.1 传统的按键去抖动程序
 - 4.2 优化后的按键去抖动程序
- 5. 示例二：通过有限状态机实现的闹钟程序
- 6. 后记

1. 什么是有限状态机

根据维基百科上的定义，有限状态机 (finite-state machine, FSM, 简称状态机) 是表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。[

为了理解这句话，假设自己还有三天就要考试，这时候就要进入紧张的备考状态，将空闲时间用在复习上。但是，为了保证足够的精力，小睡一会儿也是十分有必要的。那么，什么时候复习，什么时候睡觉呢？可以这样描述：

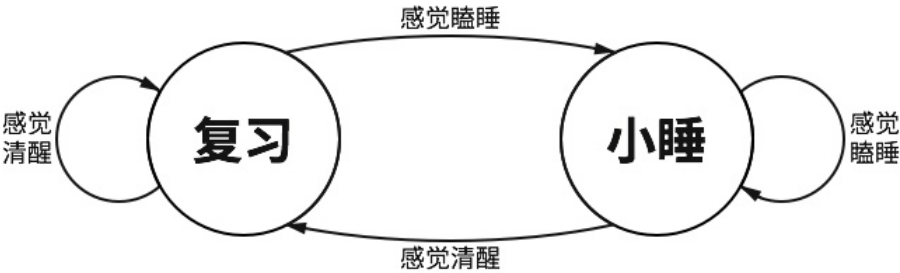
在复习的时候：

如果 感到瞌睡，则 睡觉
如果 没有感觉到瞌睡，则 继续复习

在小睡的时候：

如果 感觉不再瞌睡，则 开始复习
如果 感觉依旧瞌睡，则 继续睡觉

也可通过一幅简单的示意图（也叫「状态转移图」）表示出来：



这个例子其实就是一个简单的有限状态机，其中，复习和小睡是两个**状态**，感觉瞌睡和感觉清醒这两个**条件**可以使状态发生**转换**。[

另外，Programming Basics [网站上也提供了状态机相关的教程，用形象化的图片解释了什么是有限状态机，可[通过此链接访问](#)。

在嵌入式程序设计中，如果一个系统需要**处理一系列连续发生的任务**，或在**不同的模式下对输入进行不同的处理**，常常使用有限状态机实现。例如测量、监测、控制等控制逻辑型应用。[

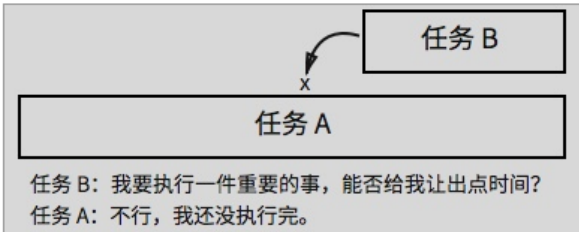
2. 有限状态机的作用

2.1 分解耗时过长的任务

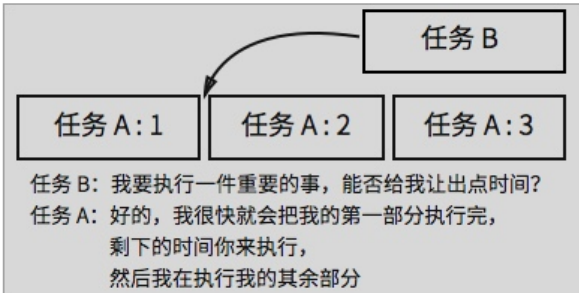
大家应该都知道，CPU 没有并行执行任务的能力。计算机「同时」运行多个程序，其实是多个程序依次交替执行，给人以程序同时运行的错觉。各个程序在什么时候开始执行，执行多长时间后切换到下一个程序，由操作系统决定。

单片机执行多任务也是类似的过程，但由于其资源有限，为了节省对 CPU 和存储空间的占用，在很多情况下没有使用操作系统。这时，**单片机中运行的各个任务必须在一定时间内主动执行完毕，才能保证下一个任务能够及时执行**。

对于一些需要长时间执行的任务，例如按键去除抖动、读取和播放 MP3 文件等，采用有限状态机的方式，**将任务划分为多个小的步骤（状态），每次只执行其中的一步**。这样，其他任务就有机会「插入」到这个任务之中，确保了各个任务都能按时执行。



任务 A 未使用状态机，其他任务不能及时执行

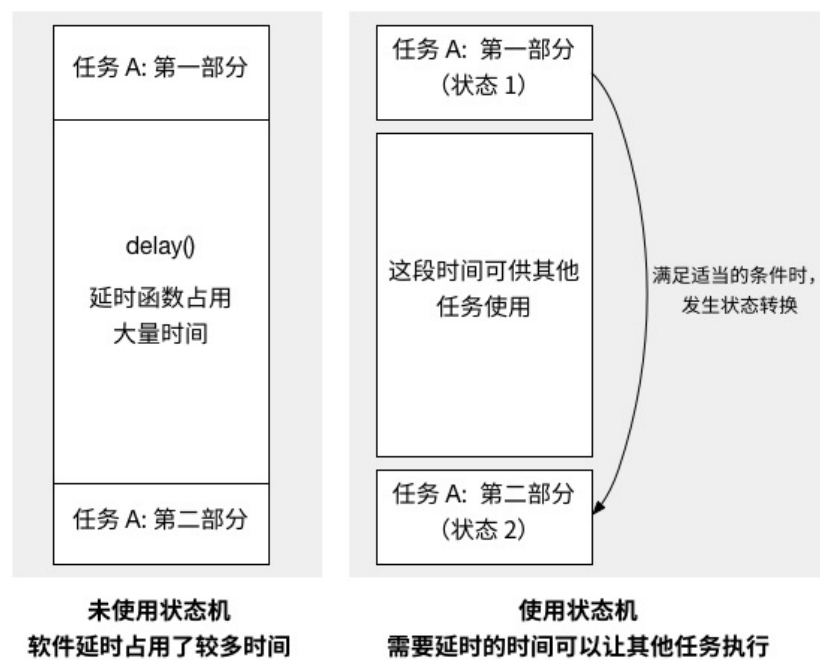


任务 A 划分为三个状态，其他任务可以及时执行

2.2 避免软件延时对 CPU 资源造成浪费

对于一些简单的程序，可通过 `delay()`, `delay_ms()` 之类的函数进行软件延时。这些延时函数，一般是通过将某个变量循环递增或递减，递增或递减到一定值后跳出循环，从而**通过消耗 CPU 时间实现了延时**。

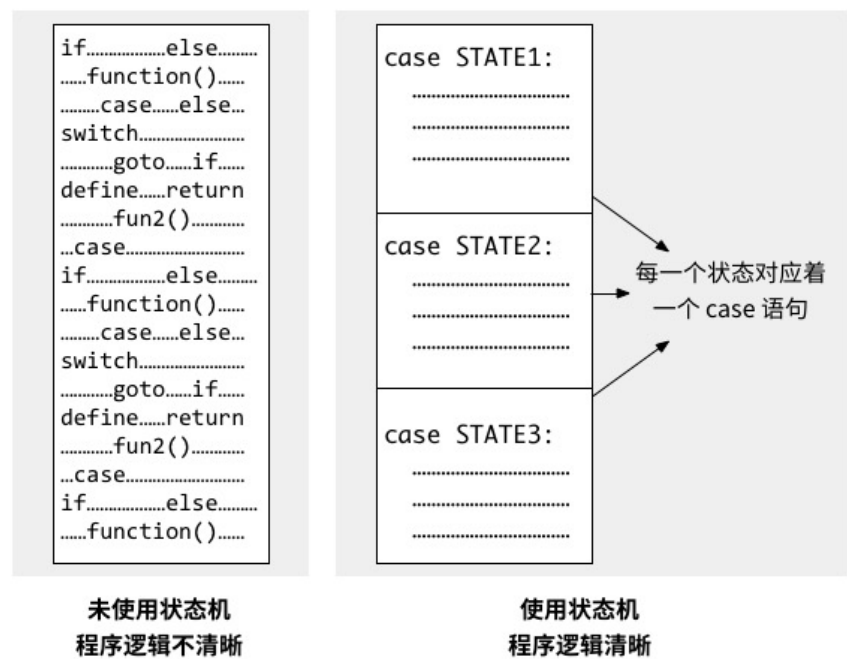
这种方式虽然简单，但在延时函数执行的过程中，其他程序无法运行，消耗了大量 CPU 资源。而通过状态机，**有助于减少软件延时的使用，提高 CPU 利用率**。



请参考下文中的 **示例一：按键去抖动程序的优化**，这一例子展示了如何通过软件延时分解耗时较长的任务，同时减少软件延时的使用。

2.3 使程序逻辑更加清晰

通过状态机，将一个复杂任务划分为多个状态，可以使程序清晰易懂，便于维护。以后想要添加、删除程序中的功能，都会变得非常容易。



下文中的 **示例二：通过状态机实现的闹钟** 展示了如何通过状态机优化程序逻辑。

3. 有限状态机的实现

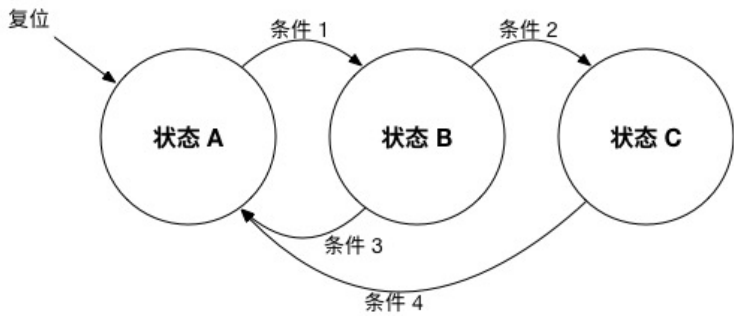
3.1 通过 switch - case 语句实现

如果使用 C 语言，switch - case 语句，即可简单地实现有限状态机。

```
01.  /* 定义各个状态所对应的数值 */
02.  #define STATUS_A 0
03.  #define STATUS_B 1
04.  #define STATUS_C 2
05.
06.  /* 该变量的值即为当前状态机所处的状态 */
07.  uint8_t currentStatus = STATUS_A;
08.
09.  /* 通过状态机实现的某个任务，
10.   * 需要放入 while(1) 等地方循环执行
11.   * /
12.  void fsm_app(void)
13.  {
14.      switch(currentStatus) /* 根据现在的状态执行相应的程序 */
15.      {
16.          case STATUS_A: /* 状态 A */
17.              doThingsForStatusA(); /* 执行状态 A 中需要执行的任务 */
18.              /* 若满足状态转换的条件，则转换到另一个状态 */
19.              if(condition_1){ currentStatus = STATUS_B; }
20.              break;
21.          case STATUS_B: /* 状态 B */
22.              doThingsForStatusB(); /* 执行状态 B 中需要执行的任务 */
23.              /* 若满足状态转换的条件，则转换到另一个状态 */
24.              if(condition_2){ currentStatus = STATUS_C; }
25.              if(condition_3){ currentStatus = STATUS_A; }
26.              break;
27.          case STATUS_C: /* 状态 C */
28.              doThingsForStatusB(); /* 执行状态 B 中需要执行的任务 */
29.              /* 若满足状态转换的条件，则转换到另一个状态 */
30.              if(condition_4){ currentStatus = STATUS_A; }
31.              break;
32.          default:
33.              currentStatus = STATUS_A;
34.      }
35.  }
```

[复制代码](#)

通过这段程序，即可实现一个具有三个状态的状态机。状态转移图如下图所示：



3.2 通过 Arduino 库实现

对于 Arduino 用户，还可以使用 FSM Library 实现。这一库将有限状态机进行了封装，可以以更简洁的方式实现状态机。
下载地址及使用说明：<http://playground.arduino.cc/Code/FiniteStateMachine>

3.3 其他方式

对于一些更复杂的任务，使用 switch - case 语句，代码会不太简洁。这时候，使用其他方式实现状态机，可能会更好。具体请查阅相关资料。

4. 示例一：按键去抖动程序的优化

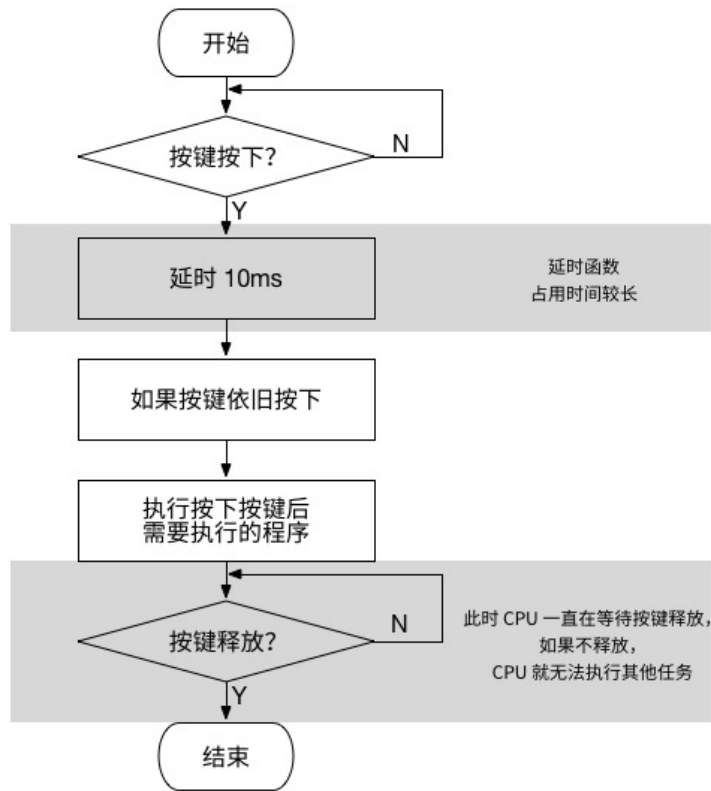
4.1 传统的按键去抖动程序

初学单片机时，我们接触的按键去抖动程序一般是这样的 [：

```
01. void keyscan()
02. {
03.     if(key1 == 0)          // 如果按键 1 按下
04.     {
05.         delays(10);        // 延时 10ms，消除因干扰产生的抖动
06.         if(key1 == 0) // 再次检测按键 1，如果依旧按下
07.         {
08.             doSomething(); //此时说明按键 1 已按下，执行按键 1 需要执行的任务
09.             while(!key1); // 等待按键释放
10.         }
11.     }
12. }
```

复制代码

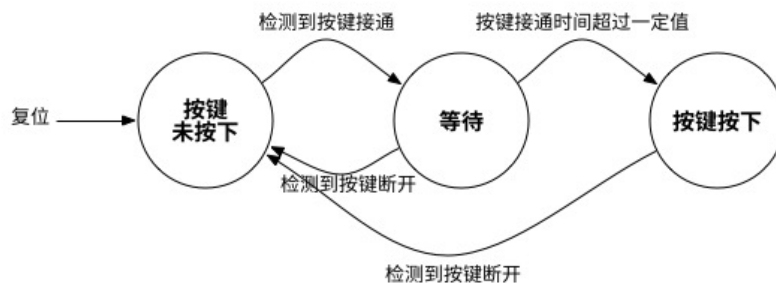
对应的流程图如下：



从流程图中可知，`delayms()` 延时函数和最后的等待按键释放的程序，会占用过多时间。

4.2 优化后的按键去抖动程序

如果使用有限状态机的思路，可以按照下图方式实现：



该状态机有三个状态，分别是**按键未按下**，**等待**，**按键按下**。当按键按下时，则会进入等待状态，若在等待状态中按键一直保持按下，说明按键已经稳定地按下，进入按键按下的状态，等待按键释放。程序代码如下：

```

01.  /* 按键去抖动状态机中的三个状态 */
02.  #define KEY_STATE_RELEASE    // 按键未按下
03.  #define KEY_STATE_WAITING   // 等待（消抖）
04.  #define KEY_STATE_PRESSED   // 按键按下（等待释放）
05.
06.  /* 等待状态持续时间
07.  * 需要根据单片机速度和按键消抖程序被调用的速度来进行调整
08.  */
09.  #define DURATION_TIME 40
10.
11.  /* 按键检测函数的返回值，按下为 1，未按下为 0 */
12.  #define PRESSED 1
13.  #define NOT_PRESSED 0
14.
15.  /* 按键扫描程序所处的状态
  
```

```
16.  * 初始状态为：按键按下（KEY_STATE_RELEASE）
17.  */
18.  uint8_t keyState = KEY_STATE_RELEASE;
19.
20.  /* 按键检测函数，通过有限状态机实现
21.  * 函数在从等待状态转换到按键按下状态时返回 PRESSED，代表按键已被触发
22.  * 其他情况返回 NOT_PRESSED
23.  */
24.  uint8_t keyDetect(void)
25.  {
26.      static uint8_t duration; // 用于在等待状态中计数
27.      switch(keyState)
28.      {
29.          case KEY_STATE_RELEASE:
30.              if(readKey() == 1) // 如果按键按下
31.              {
32.                  keyState = KEY_STATE_WAITING; // 转换至下一个状态
33.              }
34.              return NOT_PRESSED; // 返回：按键未按下
35.              break;
36.          case KEY_STATE_WAITING:
37.              if(readKey() == 1) // 如果按键按下
38.              {
39.                  duration++;
40.                  if(duration >= DURATION_TIME) // 如果经过多次检测，按键仍然按下
41.                  { // 说明没有抖动了，可以确定按键已按下
42.                      duration = 0;
43.                      keyState = KEY_STATE_PRESSED; // 转换至下一个状态
44.                      return PRESSED;
45.                  }
46.              }
47.              else // 如果此时按键松开
48.              { // 可能存在抖动或干扰
49.                  duration = 0; // 清零的目的是便于下次重新计数
50.                  keyState = KEY_STATE_RELEASE; // 重新返回按键松开的状态
51.                  return NOT_PRESSED;
52.              }
53.              break;
54.          case KEY_STATE_PRESSED:
55.              if(readKey() == 0) // 如果按键松开
56.              {
57.                  keyState = KEY_STATE_RELEASE; // 回到按键松开的状态
58.              }
59.              return NOT_PRESSED;
60.              break;
61.          default:
62.              keyState = KEY_STATE_RELEASE;
63.              return NOT_PRESSED;
64.      }
65.  }
```

[复制代码](#)

该程序也可经过扩展，实现判断按键双击、长按等功能。只需增加相应的状态和转移条件即可。

5. 示例二：通过有限状态机实现的闹钟程序

最近正在制作一个闹钟。这个闹钟支持播放 MP3 格式的闹钟声 [，支持贪睡模式，同时还有一些功能打算以后再添加上。为了使程序逻辑更加清晰，也为了方便地添加新功能，我打算采用有限状态机实现。相关程序如下：

```
01. #include "App_Alarm.h"
02. #include "USART1.h"
03. #include
04. #include "diag/Trace.h"
05.
06. /* 相关常量定义 */
07. #define ALARM_MUSIC_END 0 // 闹钟音乐播放完毕
08. #define FORMAT_OK 0 // 格式正确
09. #define FORMAT_ERROR (-1) // 格式错误
10.
11. /* 输入信息定义
12. * 作为函数的返回值供函数 getInput() 使用
13. * getInput() 将获取并返回键盘或触摸屏等设备中输入的控制命令或闹钟时间值
14. */
15. #define INPUT_ERROR (-1) // 输入格式错误
16. #define INPUT_CANCEL (-2) // 输入了「取消」命令
17. #define INPUT_SNOOZE (-3) // 输入了「小睡」命令
18. #define INPUT_ALARM_ON (-4) // 输入了「打开闹钟」命令
19. #define NO_INPUT (-10) // 没有输入
20.
21. /* 输出信息定义
22. * 作为为函数的参数供函数 displayMessege() 使用
23. * displayMessege() 用于在显示屏上显示相关的提示信息
24. */
25. #define MESSEGE_SET_ALARM_TIME (0) // 提示: 设置闹钟时间
26. #define MESSEGE_CLEAR (1) // 提示: 已取消
27. #define MESSEGE_ALARM_IS_ON (2) // 提示: 闹钟已打开
28. #define MESSEGE_WAITING (3) // 提示: 等待闹钟响起
29. #define MESSEGE_SET_SNOOZE_TIME (4) // 提示: 设置小睡时间
30. #define MESSEGE_GET_UP (5) // 提示: 该起床了
31.
32. /* 闹钟的状态 */
33. enum alarmStates
34. {
35.     ALARM_OFF, // 闹钟关闭
36.     SET_ALARM_TIME, // 设置闹钟时间
37.     WATING_FOR_ALARM, // 等待闹钟响起
38.     PLAY_ALARM_MUSIC, // 播放闹钟音乐
39.     SET_SNOOZE_TIME // 设置贪睡时间
40. } alarmState = ALARM_OFF; // 默认状态: 闹钟关闭
41.
42. /* 相关函数的定义 */
43. int16_t getInput(void);
44. void displayMessege(uint8_t);
45. void setAlarm(int16_t);
46. int16_t alarmTimeDiff(void);
47. int8_t playAlarmMusic(void);
48. void setSnooze(int16_t);
49. uint8_t checkAlarmFormat(int16_t);
50. uint8_t checkSnoozeFormat(int16_t);
51.
52. /*
53. * 闹钟主程序, 需要放入 while(1) 中循环调用
54. */
55. void alarmApp(void)
56. {
57.     int16_t input; // 输入值暂存在这个变量中
58.     switch (alarmState) // 获取闹钟状态, 下面程序将根据闹钟的状态执行相应的任务
59.     {
60.         /* 状态: 闹钟关闭
61.         * 在此状态中, 将会不断检查是否打开闹钟, 如果打开了闹钟, 则会进入下一个状态: 设置闹钟时间
62.         */
63.         case ALARM_OFF:
64.             if (getInput() == INPUT_ALARM_ON) // 检查是否打开了闹钟
65.             { // 如果打开了闹钟
66.                 displayMessege(MESSEGE_SET_ALARM_TIME); // 在屏幕或串口上提示: 请设置闹钟时间
```



```

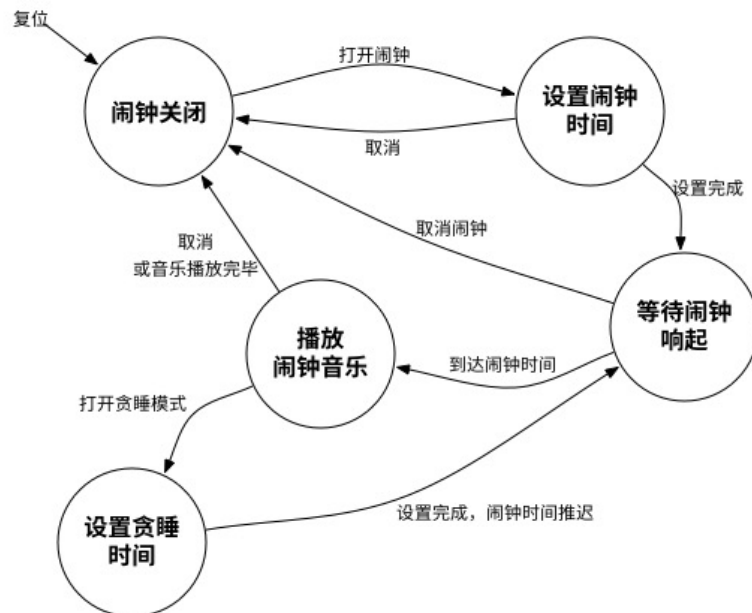
67.         alarmState = SET_ALARM_TIME;           // 进入下一个状态：设置闹钟时间
68.     }
69.     break;
70. /* 状态：设置闹钟时间
71.  * 在此状态中，将会检查输入值，
72.  * 如果
73.  *     输入“取消”命令，则取消闹钟设置，返回到闹钟关闭的状态
74.  *     输入闹钟时间格式错误，则状态不变，等待下一次重新输入
75.  *     输入了正确的闹钟时间，则设置闹钟，显示闹钟设置成功，并进入下一状态：等待闹钟响起
76.  */
77. case SET_ALARM_TIME:
78.     input = getInput();           // 获取输入值
79.     if(input == INPUT_CANCEL)     // 如果输入了“取消”
80.     {
81.         displayMessege(MESSEGE_CLEAR); // 显示“已取消”
82.         alarmState = ALARM_OFF;       // 进入状态：关闭闹钟
83.     }
84.     else if(checkAlarmFormat(input) == FORMAT_OK) // 如果输入格式正确
85.     {
86.         displayMessege(MESSEGE_ALARM_IS_ON); // 显示“成功设置闹钟，闹钟已启动”
87.         setAlarm(input); // 根据输入值设置闹钟
88.         alarmState = WATING_FOR_ALARM; // 进入下一状态：等待闹钟响起
89.     }
90.     break;
91. /* 状态：等待闹钟响起
92.  * 在此状态中，将会检查是否到达闹钟时间，如果到达，则进入下一状态：播放闹钟音乐
93.  * 同时，在此状态中也会检查输入，如果输入了“取消”的命令，则进入闹钟关闭的状态
94.  */
95. case WATING_FOR_ALARM:
96.     displayMessege(MESSEGE_WAITING); // 显示等待闹钟响起的信息，例如离闹钟响起还有多长时间
97.     if (alarmTimeDiff() <= 0) // 检查离闹钟响起还有多少时间，如果时间小于等于零（到达闹钟时间）
98.     {
99.         alarmState = PLAY_ALARM_MUSIC; // 进入下一个状态：播放闹钟音乐
100.    }
101.    if(getInput() == INPUT_CANCEL) // 如果输入了“取消”命令
102.    {
103.        displayMessege(MESSEGE_CLEAR);
104.        alarmState = ALARM_OFF;       // 进入闹钟关闭的状态
105.    }
106.    break;
107. /* 状态：播放闹钟音乐
108.  * 在此状态中，将播放闹钟音乐，若播放完毕，进入闹钟关闭的状态
109.  * 同时，在此状态中也会检查输入，
110.  *     如果输入了“小睡”的命令，则进入状态：设置小睡时间
111.  *     如果输入了“取消”的命令，则进入状态：闹钟关闭
112.  */
113. case PLAY_ALARM_MUSIC:
114.     displayMessege(MESSEGE_GET_UP); // 显示消息：“该起床了”
115.     if(playAlarmMusic() == ALARM_MUSIC_END) // 播放闹钟音乐
116.     { // 若音乐播放完毕
117.         displayMessege(MESSEGE_CLEAR);
118.         alarmState = ALARM_OFF; // 进入状态：闹钟关闭
119.     }
120.     input = getInput();
121.     if(input == INPUT_SNOOZE) // 若输入了“小睡”的命令
122.     {
123.         displayMessege(MESSEGE_SET_SNOOZE_TIME); // 显示消息：“请设置小睡时间”
124.         alarmState = SET_SNOOZE_TIME; // 进入状态：设置小睡时间
125.     }
126.     if(input == INPUT_CANCEL) // 若输入了“取消”命令
127.     {
128.         displayMessege(MESSEGE_CLEAR);
129.         alarmState = ALARM_OFF; // 进入状态：闹钟关闭
130.     }
131.     break;
132. /* 状态：设置小睡时间

```

```
133.      * 在此状态中, 将从输入获取小睡时间, 并将闹钟时间加上小睡时间, 进入状态: 等待闹钟响起
134.      */
135.      case SET_SNOOZE_TIME:
136.          input = getInput(); // 获取输入
137.          if(input == INPUT_CANCEL)
138.          { // 若输入“取消”, 则进入“闹钟关闭”的状态
139.              displayMessege(MESSEGE_CLEAR);
140.              alarmState = ALARM_OFF;
141.          }
142.          else if(checkSnoozeFormat(input) == FORMAT_OK)
143.          { // 若输入格式正确
144.              setSnooze(input); // 设置新的闹钟时间
145.              alarmState = WATING_FOR_ALARM; // 进入状态: 等待闹钟响起
146.          }
147.          break;
148.      default:
149.          displayMessege(MESSEGE_CLEAR);
150.          alarmState = ALARM_OFF;
151.      }
152.  }
```

[复制代码](#)

状态转移图如图所示：



6. 后记

在单片机编程时，如果遇到代码复杂、任务占用时间过长等问题，可以尝试通过有限状态机解决。

之前写过一个[针对 Arduino 的合作式任务调度器](#)。配合有限状态机，更有利于多任务处理。

另外，instructables 上的一篇文章通过三个实例演示了有限状态机在 Arduino 上的应用，如果感兴趣，可以通过这个链接阅读：<http://www.instructables.com/id/Arduino-Finite-State-Machine/>

备注：


1. 来源：<http://zh.wikipedia.org/zh-cn/有限状态机>
2. 为了便于理解，此处描述的状态机及状态转移图省略了一些内容，例如没有标明开始状态

- 3. Programming Basics 针对初学者的互动编程学习网站，网址为：<http://programmingbasics.org>
- 4. 根据 [Finite State Machines for MSP430 \(Rev. A\)](#) 翻译
- 5. 改编自《[新概念51单片机C语言教程](#)》中相关内容
- 6. 其实 MP3 播放程序也可以通过有限状态机实现，因为为了实现 MP3 播放持续时间较长（一首歌的时间），而且需要完成多个步骤（打开文件、读取文件、将数据发送到 MP3 解码芯片、告诉 MP3 解码芯片音乐播放完毕等）

评分

参与人数	2	+6	+5	+10	理由	收起 ▲
LEO 幻生幻灭	+ 3				赞一个!	
 lawrencedon	+ 3		+ 5	+ 10	非常详细，支持原创：)	

查看全部评分

分享到:  QQ好友和群

★ 收藏 47

 回复 举报

leicheng

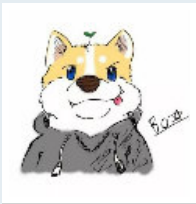



 发表于 2015-2-8 23:29:40 | 只看该作者 2#

有些原理和编译原理的知识很相似。点赞~！

 回复  支持  反对 举报

164335413



 发表于 2015-2-9 09:19:36 | 只看该作者 3#

楼主分享的教程很不错！实时操作系统的感觉，把各个任务分时处理。可以在Arduino上建立个操作系统~

 回复  支持  反对 举报

maxims



 发表于 2015-2-9 09:59:15 | 只看该作者 4#



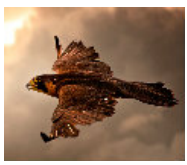



楼主好人~
先收藏，慢慢再看

 回复  支持  反对 举报

liangquan

 发表于 2015-2-9 15:34:04 | 只看该作者 5#

真没想到，还有这么高深的内容，果断收藏学习

	<div><div>回复支持反对</div><div>举报</div></div>
<div>haichaofine</div> 	<div><div><div><div></div></div>发表于 2015-2-9 22:13:54 只看该作者</div><div>6#</div></div> <div>好分享，收藏！</div> <div><div>回复支持反对</div><div>举报</div></div>
<div>openmmoo</div> 	<div><div><div><div></div></div>发表于 2015-2-10 10:50:24 只看该作者</div><div>7#</div></div> <div>原理知道，只是例子太少，在实际中有时不知道如何做！如果有更多的例子就更容易用到自己程序中了</div> <div><div>回复支持反对</div><div>举报</div></div>
<div>suoma</div> 	<div><div><div><div></div></div>发表于 2015-3-2 14:42:28 只看该作者</div><div>8#</div></div> <div>谢谢分享学习一下</div> <div><div>回复支持反对</div><div>举报</div></div>
<div>糯米基</div> 	<div><div><div><div></div></div>发表于 2015-3-4 17:22:15 只看该作者</div><div>9#</div></div> <div>楼主，写的不错{:soso_e179:}。看出了一个小问题，3.1 通过 switch - case 语句实现这部分图片，条件2和条件3是否应该对调一下，不然和程序对不上</div> <div><div>回复支持反对</div><div>举报</div></div>
<div>Blanboom</div> 	<div><div><div><div><div></div></div>楼主 发表于 2015-3-4 20:19:07 只看该作者</div><div>10#</div></div><div><div><div>“糯米基 发表于 2015-3-4 17:22 楼主，写的不错。看出了一个小问题，3.1 通过 switch - case 语句实现这部分图片，条件2和 ...”</div></div><div>谢谢提醒，已修改</div><div><div>回复支持反对</div><div>举报</div></div></div></div>
<div>五哥U五哥</div>	<div><div><div><div></div></div>发表于 2015-3-19 15:58:54 只看该作者</div><div>11#</div></div> <div>今天在阿莫上看高了马潮老师的状态机按键消抖，，，跟这个思路一模一样。。。学习了。</div>

	<div>回复 支持 反对</div> <div>举报</div>
<div>microplc</div> 	<div>发表于 2015-4-14 21:28:16 只看该作者12#</div> <div>拜读领教了。</div> <div>回复 支持 反对</div> <div>举报</div>
<div>edyd</div> 	<div>发表于 2015-4-15 23:03:16 只看该作者13#</div> <div>好，学习了！顶一个！</div> <div>回复 支持 反对</div> <div>举报</div>
<div>zhji</div> 	<div>发表于 2015-5-5 09:56:33 只看该作者14#</div> <div>楼主最新版那个库的链接打不开，http://arduino-info.wikispaces.com/HAL-LibrariesUpdates 可否发一份。多谢了！ zhji@stu.zzu.edu.cn</div> <div>回复 支持 反对</div> <div>举报</div>
<div>zjyylj</div> 	<div>发表于 2015-5-22 13:58:32 只看该作者15#</div> <div>有点像PLC编程的思路，有趣</div> <div>回复 支持 反对</div> <div>举报</div>

下一页 »

发帖

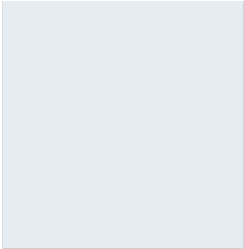
返回列表121 / 2 页下一页

B A    

高级模式

https://www.geek-workshop.com/thread-12693-1-1.html

13/14



您需要登录后才可以回帖 登录 | 注册  用QQ帐号登录

[发表回复](#) ☐ 回帖后跳转到最后一页

[本版积分规则](#)