目录

[摘要 4](#_Toc451891867)

[关键字： 4](#_Toc451891868)

[ABSTRACT 5](#_Toc451891869)

[1 引言 6](#_Toc451891870)

[1.1 课题背景 6](#_Toc451891871)

[1.2 课题任务 6](#_Toc451891872)

[1.3 论文结构 7](#_Toc451891873)

[2 背景知识 8](#_Toc451891874)

[2.1 MSP430F149微处理器 8](#_Toc451891875)

[2.1.1 主要特性 8](#_Toc451891876)

[2.1.2 具体功能 9](#_Toc451891877)

[2.2 状态机 9](#_Toc451891878)

[2.3 智能串口屏 10](#_Toc451891879)

[2.3.1 HMI介绍 10](#_Toc451891880)

[2.3.2 串口HMI介绍 10](#_Toc451891881)

[2.4 云台 10](#_Toc451891882)

[2.4.1 云台简介 10](#_Toc451891883)

[2.4.2 云台控制协议 11](#_Toc451891884)

[3 系统整体介绍 13](#_Toc451891885)

[3.1 输入 13](#_Toc451891886)

[3.2 输出 13](#_Toc451891887)

[3.3 数据处理 13](#_Toc451891888)

[3.4 整体设计 14](#_Toc451891889)

[3.4.1 系统组成 14](#_Toc451891890)

[3.4.2 功能设计 14](#_Toc451891891)

[3.4.3 操作流程 15](#_Toc451891892)

[4 系统硬件设计 17](#_Toc451891893)

[4.1 硬件组成 17](#_Toc451891894)

[4.2 最小系统电路设计 17](#_Toc451891895)

[4.3 电源管理电路设计 17](#_Toc451891896)

[4.4 时钟电路 19](#_Toc451891897)

[4.5 485通讯电路 19](#_Toc451891898)

[4.6 按键电路设计 19](#_Toc451891899)

[4.7 继电器电路设计 20](#_Toc451891900)

[5 系统软件设计 22](#_Toc451891901)

[5.1 软件分析 22](#_Toc451891902)

[5.1.1 需求概述 22](#_Toc451891903)

[5.1.2 数据流分析 23](#_Toc451891904)

[5.1.3 数据流图 23](#_Toc451891905)

[5.2 程序设计 25](#_Toc451891906)

[5.2.1 程序流程图 25](#_Toc451891907)

[5.3 用户输入 27](#_Toc451891908)

[5.3.1 按键 27](#_Toc451891909)

[5.3.2 旋钮 29](#_Toc451891910)

[5.4 显示 31](#_Toc451891911)

[5.4.1 显示屏选择 31](#_Toc451891912)

[5.4.2 显示屏资源 32](#_Toc451891913)

[5.4.3 显示框架设计 33](#_Toc451891914)

[5.4.4 页面 34](#_Toc451891915)

[5.5 时间模块 36](#_Toc451891916)

[5.5.2 DS1302介绍 36](#_Toc451891917)

[5.5.3 时间接口 41](#_Toc451891918)

[5.6 数据保存 42](#_Toc451891919)

[5.6.1 数据保存介质 42](#_Toc451891920)

[5.6.2 flash介绍 43](#_Toc451891921)

[5.6.3 flash操作驱动 43](#_Toc451891922)

[5.7 控制输出 44](#_Toc451891923)

[5.7.1 方向控制 44](#_Toc451891924)

[5.7.2 灯光控制 45](#_Toc451891925)

[5.8 软件开发环境 46](#_Toc451891926)

[6 运行结果 49](#_Toc451891927)

[6.1 装置部件 49](#_Toc451891928)

[6.2 自动模式 50](#_Toc451891929)

[6.3 手动模式 51](#_Toc451891930)

[7 毕业设计总结 52](#_Toc451891931)

[致谢 53](#_Toc451891932)

[参考文献 54](#_Toc451891933)

[附录 55](#_Toc451891934)

[附录1 部分程序代码 55](#_Toc451891935)

# 摘要

本设计目的为设计一个可以辅助人们管理植物的装置，随着社会的发展，自动管理植物的需求将会越来越大，本装置能够代替人们来管理植物，包括补光、浇水等管理功能，其中补光可以设置不同的光谱组合，并且可以同时管理多种植物。

整个系统可以分为输入、数据处理、输出三个逻辑模块，输入部分包括用户的数据输入、传感器输入等，输出为执行部分的控制输出，以及对用户输出的状态信息。数据处理主要为将输入数据、当前时间、当前位置进行处理得出输出数据。整个系统既可以自动按照计划来执行管理功能，也可以通过用户来手动控制实时的输出，既可以满足自动化要求，也可以满足多样性的要求。

通过此植物管理辅助装置，增加了植物管理的自动化水平，管理植物不再依靠人们自己花费精力来照料，避免了人为疏忽对植物造成的影响，同时解放人力，减少人们的劳动时间。本设计使用方便、运行稳定，为植物的自动化管理提供了一个较好的解决方案。

关键字：植物管理、MCU、按键、显示、时间、flash

# ABSTRACT

The design for the purpose of designing a device that people can assist the management of plants, with the development of society, the demand for automatic management of the plant will be increasing, the device can replace the people to manage the plant, including the fill light, watering and other management functions wherein the fill light can be set to different spectral composition, and can manage a variety of plants.

The whole system can be divided into input, data processing, three output logic module, an input section includes user data input, sensors and other input, the output is the control output of the operative part, and the output state of the user's information. Mainly for data processing of the input data, the current time, the current positions that deal with the output data. The entire system can be automatically performed in accordance with the management plan, which can also be manually controlled by the user real-time output, both to meet the automation requirements, but also to meet the diverse requirements.

This plant management assistive devices, increasing the level of automation of the plant management, plant management no longer rely on one's own energy to take care to avoid the impact of the plant caused by human negligence, while the liberation of manpower, reducing people's work time. This design easy to use, stable operation and management of plant automation provides a better solution.

**Keywords：**plant management, MCU, keys, display, time, flash

# 1 引言

## 1.1 课题背景

随着人们生活的发展，对生活环境要求的提高，可以改善人们家居环境的各种植物走进了日常中，因此照料植物也成为了需要解决的一个问题，本设计的目标就是实现一个可以帮助人们管理植物的辅助装置。

植物的管理主要的方面有补光和浇水，因此从这两个方面出发，去设计一个能够实现自动补光和浇水功能的装置，以代替人力方便人们的管理。在补光时，选择LED植物补光灯来控制光照，植物的光合作用需要的光线，波长在400~720nm左右。440~480nm（蓝色）的光线以及640~680nm（红色）对于光合作用贡献最大。520~610（绿色）的光线，被植物色素吸收的比率很低。因此可以据此确定实际补光时选择的光源。

## 1.2 课题任务

设计一种植物辅助管理装置，以用于在生活中帮助人们管理植物。在功能方面，主要实现的有浇水和补光的辅助管理、同时管理多种不同的植物、自动和手动两种不同的控制模式。从要实现的功能、交互方式、控制方式几个方面入手，本设计以微控制器为核心，通过数据采集、数据处理、数据存储、输出执行这几个过程，实现了对植物的辅助管理，如图1-1，为该装置在实际中的应用框图。



图 1‑‑1 植物管理辅助装置应用框图

## 1.3 论文结构

本论文将依照章节顺序依次介绍该设计的各个实现。

第一章，引言，以引出下文。

第二章，列出所需的背景知识，主要为本设计所主要用到的器件、知识点。

第三章，系统整体上的介绍，在系统等级描述本设计。

第四章，系统的硬件部分设计实现，内容为系统中各个模块硬件设计的电路原理图和环境的介绍。

第五章，软件设计，详细描述系统中软件设计部分重要部分的设计与实现。包括软件开发的环境，以及按键模块，旋钮模块，显示模块、时间模块、计划模块等模块的介绍。

第六章，调试，介绍开发中的调试环境，以及重要的测试过程和最终结果。

第七章，总结。

# 2 背景知识

## 2.1 MSP430F149微处理器

MSP430F149微处理器是德州仪器（TI）公司的 MSP 超低功耗 MCU 系列的 MSP430F1x 微控制器 (MCU) 中的一个，广泛适用于消费类电子产品、数据记录应用、便携式医疗仪器等的 16 位微控制器。如图2-1，为MSP430F1xx的结构框图。

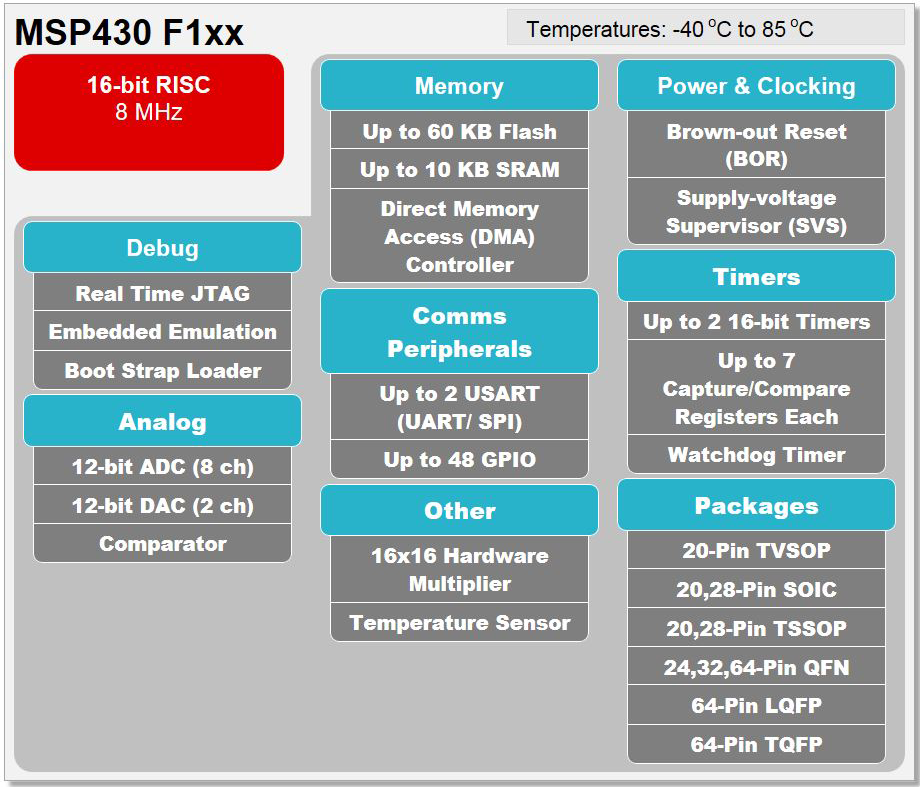


图 2-1 MSP430F1xx 结构框图

### 2.1.1 主要特性

MSP430F1x MCU系列微处理器有如下通用属性：

* 高达 8MHz 的 CPU 速度高达 60KB 闪存
* 高达 10KB SRAM
* 1.8-3.6V 工作电压
* 广泛适用于各种高性能模拟和智能数字外设
* 200μA/MHz 工作模式
* 0.7μA 实时时钟模式
* 0.1μA RAM 保持模式
* 在 6μs 之内从待机模式超快唤醒

### 2.1.2 具体功能

#### a. 存储

存储器分为2KB ram，60KB main flash，256B information flash

#### b. 定时

定时器相关的模块有三个16位的定时器，一个看门狗，其余两个为通用定时器，带捕捉、比较、pwm功能等。

#### c. 串行通信

有两个通用同步异步收发机USART，每一个可分别配置为uart、spi功能。

#### d. 模拟量处理

一个12位的ADC，一个模拟电压比较器。

#### e. gpio

6个8位的数字io端口。

## 2.2 状态机

在软件编程中，状态机是一个经常用到的模型，在处理多状态情况时有很大的优点。用一个确切的描述来描述状态机，状态机是一个有向图形，从组成上描述为一组节点和一组对应的转移函数，从动作上描述是通过响应一系列事件而运行。每个事件都在属于“当前”节点的转移函数的控制范围内，其中函数的范围是节点的一个子集。函数返回“下一个”（也许是同一个）节点。这些节点中至少有一个必须是终态。当到达终态，状态机停止。状态机是一种概念性机器，它能采取某种操作来响应一个外部事件。具体采取的操作不仅能取决于接收到的事件，还能取决于各个事件的相对发生顺序。之所以能做到这一点，是因为机器能跟踪一个内部状态，它会在收到事件后进行更新。为一个事件而响应的行动不仅取决于事件本身，还取决于机器的内部状态。另外，采取 的行动还会决定并更新机器的状态。这样一来，任何逻辑都可建模成一系列事件/状态组合。

## 2.3 智能串口屏

### 2.3.1 HMI介绍

HMI是Human Machine Interface的缩写，中文为“人机接口”，也可以说是人机界面。它是系统和用户之间交互和信息交换的媒介，信息的内部形式通过它转换为人类可以接受的形式。只要是有这人机信息交流的地方，都存在着人机界面。

### 2.3.2 串口HMI介绍

串口HMI是设备封装好 HMI 的底层功能以后，通过串口(USART 232)与用户MCU进行交互，比如MCU可以随时通过USART发指令通知设备切换某个页面或者改变某个组件的属性。设备也可以随时通过 USART 通知用户 MCU 操作者目前触摸了页面上的某个组件或者设备当前进入了某个页面。对于产品研发者来说，产品开发初期可以选型的接口一般有3种类型：RGB接口，MCU总线接口,串口HMI。

RGB接口必须用在带有RGB驱动的ARM芯片上；MCU总线接口驱动比 RGB简单一些，对硬件也基本没有任何要求，但是显示速度和界面的显示驱动不是很理想；串口HMI是最简单的显示方案，跟MCU总线屏一样对用户的硬件没有任何要求，界面的显示是设备内部自己实现的，没有速度瓶颈，用户MCU只是发送指令，并不需要底层驱动。针对显示的人机界面的布局和大多数的逻辑（比如界面背景，按钮效果，文本显示等）不需要用户的MCU参与，使用和设备一同提供的上位软件，在电脑上可以直接预先制作好界面设计。串口HMI屏的缺点是硬件的生产成本要比较高，并且由于是用户MCU间接的控制显示屏，响应速度回比较慢。

## 2.4 云台

### 2.4.1 云台简介

云台是安装、固定摄像机的支撑设备，它分为固定和电动云台两种。 固定云台适用于监视范围不大的情况，在固定云台上安装好摄像机后可调整摄像机的水平和俯仰的角度，达到最好的工作姿态后只要锁定调整机构就可以了。电动云台适用于对大范围进行扫描监视，它可以扩大摄像机的监视范围。

### 2.4.2 云台控制协议

监控行业通信协议应用最广的协议就是美国PELCO-P,PELCO-D协议，可以说国内大部分控制设备，包括嵌入式录像机，监控视频采集卡软件，各种控制键盘，PC机调试研究均有此协议。下面详细叙述PELCO-D协议。

PELCO-D协议一般用于矩阵和其它设备之间的通信。它的格式如下：

PELCO-D协议一般用于矩阵和其它设备之间的通信。它的格式如下：

图 2-2 PELCO-D协议格式

所有的值都是用的 16进制表示。同步字通常都是 $FF。地址码是指与矩阵通信的那台设备的逻辑地址，可以在设备中设置。命令字 1和命令字2设置如下图2-3所示。

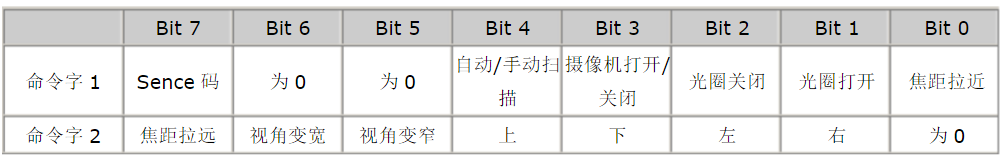


图 2-3 命令字1和命令字2设置

Sence码与Bit4和Bit3有关。在Bit4和Bit3为1的情况下，如果Sence码为1，则命令就是自动扫描和和摄像机打开；如果Sence码为0，则命令就是手动扫描和摄像机关闭。当然如果Bit4或Bit3为0的话那命令就无效了。数据1表示镜头左右平移的速度，数值从$00(停止)到$3F(高速)，另外还有一个值是$FF，表示最高速。数据2表示镜头上下移动的速度，数值从$00(停止)到$3F(最高速)。校验码是指Byte2到Byte6这5个数的和(若超过255则除以256然后取余数)。

图2-4列出了pelco-D协议主要的控制命令，如下所示。



图 2.4 pelco-D协议主要控制命令

# 3 系统整体介绍

## 3.1 输入

系统的运行是围绕着外部输入数据进行的。外部数据的输入分两种，一种是需要用户参与的，一种是不需要用户参与的。在用户参与的时候，输入方式的选择依据两个方面，一是需要用户输入的数据类型，二是如何输入方便且容易实现。

在实际操作中，主要的操作为设置条目的值，需要通过输入选择不同的条目，并且改变选中条目的值。因此，关于输入的数据类型有两种，一是类似逻辑类型的，即输入情况少，可选范围小；另一种是可调范围比较大的。

结合使用和实现的方便容易，选择了按键和旋钮这种输入方式，按键主要做逻辑性的输入，旋钮主要做持续性的输入。

在不需要用户参与的输入中，主要依靠传感器以及对应的接口来实现数据的采集和输入。

## 3.2 输出

装置的运行结果会通过输出数据来表现出来，输出部分是直接呈现到用户的感官中的，输出可以分为控制输出部分和信息输出部分。

控制性输出包括补光控制、浇水控制和方向控制，补光控制通过开关不同光谱的灯来实现，浇水控制通过控制水泵的开关来控制浇水，在方向控制上，使用了云台来方便方向控制的实现。

## 3.3 数据处理

数据处理主要包括从输入到输出的装换，以及数据处理的中间过程。在从输入到输出的转换过程中，最主要的任务是，将用户通过输入模块输入的计划数据按计划正确的输出到控制部分。在数据处理的中间过程中，包括时间数据的维护，以及需要掉电保存的数据和flash之间的交换过程。

## 3.4 整体设计

### 3.4.1 系统组成

根据数据的处理，程序主要有输入、数据处理、输出三个部分，基于硬件详细分为按键、旋钮、显示、时间、计划处理、数据保存、模拟定时、方向控制、调试、浇水等这几个模块。

并没有在硬件模块的基础上再次做逻辑层次上的划分，这是一个有待改进的地方。系统的结构组成图如图3-1所示。



图 3-1 系统结构组成图

### 3.4.2 功能设计

装置的结构组成以及每一部分的功能用途，设计如下。

* MSP430F149，用于处理输入输出信号
* 时钟芯片DS1302，确定时间
* 按键B1~B8，用于输入信号
  + B1，切换手动和自动模式
  + B2~B4，在手动模式下控制三个光照搭配
  + B5~B6，复用，在自动模式下上下循环选择所有可以输入的条目，或者可以改变选定的条目，在手动模式下两个按键用于控制方向相关。
  + B7，复用，在自动模式下返回上一级，在手动模式下用于控制方向相关。
  + B8，在自动模式下进入所选的条目。
* 旋钮K1，在自动输入模式下用于调整当前所选的条目的值。
* 显示屏LCD1，用于显示设计的页面内容。
* led L1，表示是否正常工作，即电源是否连接
* 继电器R1~R3，用于控制执行装置
  + R1，控制红光的开关
  + R2，控制蓝光的开关
  + R3，控制UVB光的开关
* 云台S1~S2
  + S1，用于控制水平平面上的旋转
  + S2，用于控制垂直平面上的旋转

### 3.4.3 操作流程

这一节叙述了装置正常工作的操作流程。分为自动控制和手动控制两种模式，它们各自的流程介绍如下文所述。

#### a.自动控制

* 开启电源，将自动加载上一次的计划数据，并且显示屏进入主界面。
* 按B1，选择自动模式
* 如果时间不正确，使用B5~B6选择时间条目，按下B8，进入时间设置状态，完成时间设置，按B7返回。
* 使用B5~B6选择一个对象条目，按B8进入对象浏览页面，使用B5~B6选择对象对应的属性，按B8进入编辑状态，使用旋钮K1或者B5~B6编辑对应值，完成编辑按B7返回。
* 重复直到完成所有的设置对象的任务。
* 返回主界面，完成设置。

#### b.手动控制

* 开启电源，将自动加载上一次的计划数据，并且显示屏进入主界面。
* 按B1，选择手动模式。
* 按B5或B6，驱动云台旋转调整方向。
* 按B2~B4，选择合适的光照搭配。
* 完成。

# 4 系统硬件设计

## 4.1 硬件组成

本设计的硬件部分由TI公司的MSP430F149处理器、电源管理模块、时钟电路模块、485通信电路、继电器电路、按键和旋钮电路等部分组成。

## 4.2 最小系统电路设计

本设计选择的微处理器为MSP430F149，保证其正常工作需要提供最下系统电路，设计图如下图4-1所示。



图 4-1 最小系统

## 4.3 电源管理电路设计

本装置中需要12V的直流电和220V的交流电，所以需要一个交流直流转换部分。采用HLK-PM01转换模块进行转换。HLK-PM01实物如下图4-2所示。



图 4-2 HLK-PM01实物图

HLK-PM01输入特性如下表4-1所示。

表 4-1 HLK-PM01输入特性

|  |  |  |
| --- | --- | --- |
| **项目名称** | **技术要求** | **单 位** |
| 额定输入电压 | 100-240 | VAc |
| 输入电压范围 | 90-264 | VAc |
| 最大输入电流 | ≤0.2 | A |
| 输入浪涌电流 | ≤10 | A |
| 最大输入电压 | ≤270 | VAc |
| 输入缓启动 | ≤50 | mS |
| 输入低电压效率 | Vin=110VAc，输出满载≥69 | % |
| 输入高电压效率 | Vin=220VAc，输出满载≥70 | % |
| 长期可靠性 | MTBF≥100，000 | h |
| 短时间最大输出电流 | ≥1000 | mA |
| 输出纹波及噪音 | ≤50 | mV |

HLK-PM01交流直流转换电路如图4-3所示.

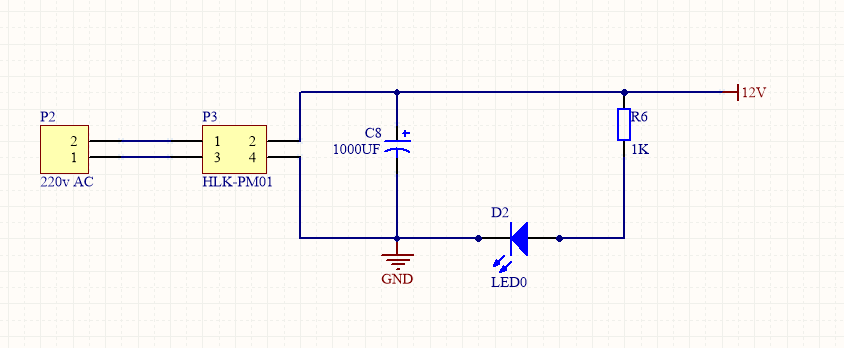


图 4-3 HLK-PM01交流直流转换电路

## 4.4 时钟电路

本装置需要实时时间，利用DS1302来进行时间处理，可以输出输入年、月、日、周、时、分、秒格式的时间，其电路如图4-4所示。



图 4-4时钟电路

## 4.5 485通讯电路

本装置使用的云台的通讯接口是485协议，选择MAX485芯片来实现此功能， 485通讯电路设计如下图4-5所示。



图 4-5 485通讯电路

## 4.6 按键电路设计

本装置在用户输入数据的时候需要按键操作，按键部分的电路设计如下图4-6所示。

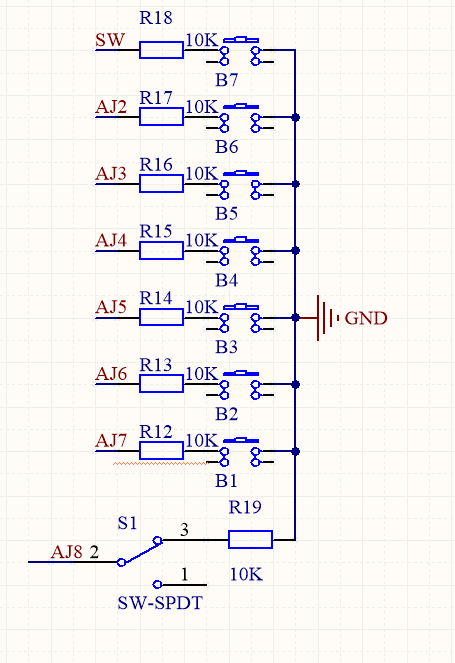


图 4-6 按键电路

## 4.7 继电器电路设计

在控制执行的时候，需要继电器来输出控制信号，继电器的电路设计如下图4-7所示。



图 4-7 继电器电路

# 5 系统软件设计

## 5.1 软件分析

### 5.1.1 需求概述

对于此植物辅助管理装置，需要完成的功能有：

* 对不同的植物设置光照、浇水周期计划，并尽量自动化。
* 可以手动控制执行装置。
* 异常处理。

下面详细说明各个功能。

#### a. 设置计划及其自动化控制

计划指不需要人控制时，装置运行所依照的预定流程。关于计划，有两个方面：

* 计划包含的元素
* 计划输入的方法

包含的元素是和植物自身特性相关的。植物需要光照与浇水这两个最基本的要求，光照要求由周期和每个周期中持续的光照时间组成。由于一个光照分为不同的光谱，所以需要记录每一种植物的光谱设置，即哪一种光。由于一个装置需要管理多种植物，所以需要记录每一个植物的位置。综上，需要记录的有光照周期、光照周期中的持续光照时间、光谱成分、位置值。浇水要求记录所需要的土壤湿度。

关于输入方法，最基本的要求是方便操作。包括操作动作实施的方便，以及操作逻辑的清晰明了。

自动化控制是对计划的执行过程。目标是尽量减少人为的干预，在保证正常管理植物的同时减少人的劳动。

#### b. 手动控制

手动控制存在的意义是为了满足多样化的管理需求。对手动控制的要求是，使用者可以通过外部输入，即时的控制执行装置，被控制的应包括计划输入的对应元素，控制的输入方法要求同计划输入的要求。也就是说，此时，管理者代替了设置计划及自动化控制这部分功能。

#### c. 异常处理

异常状态指不符合计划状态或手动控制状态要求的状态，主要包括电源故障。

### 5.1.2 数据流分析

在需求分析阶段，需要完成的一个任务是画数据流图。数据流图简称DFD，它从数据传递和加工的这两个角度，以图形方式来表达系统的逻辑功能、数据在系统内部的逻辑流向和逻辑变换过程。

对于整个系统，主要的数据包括用户的输入数据，传感器采集到的数据，设备向用户反馈的数据，设备的控制输出数据。需要完成的加工就是将用户的输入数据转换为设备的控制输出数据，并且同时产生反馈给用户的数据。在这个过程中，主要有io数据，按键数据，旋钮数据，计划数据，时间数据，环境数据，显示数据，控制数据等数据存在。

### 5.1.3 数据流图

这一节主要为系统的数据流图，下面分别介绍了数据流图中的各个层。

顶层图主要表明了系统的范围，以及与外部环境的数据交换关系。如下图5-1为系统的数据流图的顶层图。



图 5-1 顶层图

0层图对顶层图中的加工进行细化，如下图5-2为系统的数据流图的0层图。



图 5-2 0 层图

1层图，对父层数据流图的某个加工进行细化，如下图5-3分别为系统的数据流图的1层图。



图 5-3 1层图

## 5.2 程序设计

### 5.2.1 程序流程图

在没有操作系统的软件系统中，最常用的是前后台程序结构。前后台程序结构是由主函数中循环执行的任务和在中断中执行的任务组成，主函数中的任务一般不被外界干扰，称为后台程序，位于中断中的任务一般需要外部的触发，称为前台程序。

整个系统分为几个异步的任务，分别属于不同的前后台任务，后台任务主要完成时间更新、计划处理、按键事件触发等，前台任务有三个，分别完成模拟定时器实现、按键事件检测、旋钮检测等功能。它们各自的程序流程图如下所述。

下图5-4为后台任务流程图。



图 5-4 后台任务流程图

各前台任务流程图如下图5-5所示。



图 5-5 前台任务流程图

## 5.3 用户输入

### 5.3.1 按键

按键这一模块提供得到用户对键操作的接口函数。关于用户对键的操作，一共有4种，如下所列：

* 没有键操作
* 单击按键
* 双击按键
* 长按按键

这个模块提供的主要接口函数有：

void key\_detect(void);

key\_mean get\_key\_mean(key\_type key\_t);

void clear\_key\_m(void);

key\_detect函数需要每个10ms定时调用来实现按键的实时监测，get\_key\_mean函数实现了获取按键操作的接口，clear\_key\_m函数实现了清除当前按键操作的接口。下面详细叙述实现细节。

按键模块整个实现的框架是基于状态机的。此模块的功能是将外部输入的电平信号，转换为按键操作。在一个系统中按键的操作是随机的，因此系统软件中要对按键进行循环查询。在按键检测过程中需要进行消抖处理，消抖的延时处理一般要10ms或20ms，因此取状态机的时间序列为10或20ms，这样不仅可以跳过按键消抖的影响，同时也远小于按键0.3-0.5S的稳定闭合期，不会将按键过程丢失。在稳定闭合之后，可以通过计数来判断是单击、双击或者长按。

状态机的状态变量类型为：

/\*

\* key\_state - 表示按键的状态

\* key\_state\_0 初始状态，或者表示第一次单击状态

\* key\_state\_1 消抖和确认状态，或表示第二次单击状态

\* key\_state\_2 按键计时状态

\* key\_state\_3 按键等待释放状态

\*/

typedef enum key\_state\_

{

key\_state\_0 = 0, key\_state\_1, key\_state\_2, key\_state\_3

} key\_state;

首先，用一个状态机模型完成从io电平状态到无键、单击、长按这三种操作的转换。在这个状态机里，初始状态为key\_state\_0，设按键按下对应io电平状态为有效，没有按下对应为无效，下面说明不同的状态下对应的处理过程。

* 在key\_state\_0状态下，判断io电平是否为有效，如果有效，切换到key\_state\_1状态。
* 在key\_state\_1状态下，判断io电平是否有效，如果有效，表明消抖完成，确实有键按下，因此初始化计数变量为0，并且将状态切换到key\_state\_2状态，如果无效，表明这是一次抖动，将状态切换到key\_state\_0就可以了。
* 在key\_state\_2状态下，判断io电平是否有效，如果无效，设返回按键操作为单击，同时状态切换到key\_state\_0，如果有效，首先给计数变量加1，判断计数变量是否大于100，如果是大于100的话，设返回按键操作为长按，同时状态切换到key\_state\_3，以等待按键释放，如果不大于，不做处理。
* 在key\_state\_3状态下，判断io电平是否有效，如果无效的，状态切换到key\_state\_0，如果有效，不做处理。

上面完成了一个无键、单击、长按这三种操作的检测，下面以这个状态机为基础，使用另一个状态机完成无键、单击、双击、长按这四种操作的检测。同样，在这个状态机里，初始状态为key\_state\_0，下面说明不同的状态下对应的处理过程。

* 在key\_state\_0状态下，判断是否有单击操作，如果有，则需要判断后续事件，将计数变量清零，设置返回按键操作为无键，并且将状态切换到key\_state\_1，如果不是单击操作，则返回原操作。
* 在key\_state\_1状态下，判断是否有单击操作，如果有，设置返回按键操作为双击，并将状态切换到key\_state\_0，如果没有，将计数变量加1，判断计数变量是否大于30，如果大于的话，说明时间距上一次单击操作300ms时间内没有出现另一次单击操作，设置返回按键操作为单击，并将状态切换到key\_state\_0。

### 5.3.2 旋钮

旋钮模块提供得到用户对旋钮的旋转量的接口。

旋钮使用的是一个增量式编码器，编码器有绝对式和增量式这两种，在位置和角度的测量中有广泛的应用。绝对式编码器的优点是其输出信号和旋转位置之间存在一一对应的关系，但是实现中其测量精度相对不高，所花费的成本相对较高；增量式编码器是一个将相对旋转位移转换为脉冲个数的器件，输出信号为数字脉冲信号，如果在程序中加上一个设定的基准位置的话，可以将相对的脉冲个数转换为绝对的位置量，实现绝对式测量绝对位置的功能，并且成本相对较低，精度也较高。

增量式编码器在输出方式上也可以分为不同的类型，主要有差分式和非差分式两种，信号形式都是脉冲信号，不同的是，差分式的输出有两相，A和B，并且输出高电平接近工作电源电压；而茶饭输出有四个不同的相，A，A非，B，B非，其中A和A非之间是反相的关系，B和B非之间是反相的关系，并且其输出的高电平在工作电源电压的半数附近，差分就是体现在这儿。需要说明的是，不管是哪种形式，A与B相的波形在形式上是完全相同的，不同的仅仅是在相位上存在90度的相差。对于所使用的增量式编码器，其输出的典型脉冲波形如图5-6所示。



图 5-6 增量式编码器输出波形

旋钮旋转只有逆时针与顺时针2个旋转方向，向逆时针方向旋转对应着图中从左到右的脉冲输出波形，向逆时针方向旋转对应着图中从左到右的脉冲输出波形。

如果增量式编码器顺时针旋转，则A相滞后B相90度；如果逆时针旋转，则B相滞后A相90度，而且编码器各相的输出电平完全取决于其旋转的位置。编码器旋转一周，A相与B相所输出的脉冲数相同，其脉冲数决定了编码器的精度，通过从编码器读取脉冲数，则可以计算出其绝对位置，如果有一个预置的位置，则可以计算出其绝对位置。

当B相为高电平是，如果检测到A相有一个上升沿，则可以说明波形从左向右云顶，则令计数器加1,；当B相为低电平是，如果检测到A相有一个上升沿，则可以说明波形从右向左运动，则令计数器减1。

上边所述测量方法的前提是在A相和B相的波形中，在B相的一个稳定电平期间，A相的波形有且只能有一个沿跳变，但是，在实际操作中，这个条件不一定被满足，对于一个A相的沿跳变位置处，可能会在这个位置处来回抖动，这样会产生多于一个的沿跳变，计数器会在一个方向上一直累计，但是实际并没有旋转，这样就产生了读取误差，并且这个误差量是和所用的编码器的精度没有关系，实质是由测量方法的不当造成的。

为了解决上述问题，需要改进测量方法。上述方法主要的问题是将上升沿和下降沿没有区分开来，导致方向的确定仅仅有B相的高低电平决定，所以改进的思路主要将上升沿可下降沿加入到方向确定中来，有图中可以看到，对于B相的一个稳定的高电平期间，当波形从左往右时，A相对应期间只有上升沿，当波形从右往左是，A相对应期间只有下降沿。B相的低电平期间也是同样的过程，这样就避免了不能区分沿跳变，即避免了测量方法造成的误差。旋钮模块就是依据此测量方法实现的。

根据以上论述实现旋钮的输入检测功能，按模块提供的接口有：

void knob\_enable(void);

void knob\_disable(void);

void knob\_clear(void);

int32\_t get\_knob\_val(void);

void knob\_detect(void);

其中，knob\_enable函数用于使能旋钮的输入检测；knob\_disable函数用于禁止旋钮的输入检测；knob\_clear函数用于清除旋钮当前的输入值，即将重置相对位置；get\_knob\_val函数用于得到当前旋钮的输入值；knob\_detect这个函数是旋钮的前台执行部分，是旋钮的主要功能实现部分。

## 5.4 显示

### 5.4.1 显示屏选择

显示屏选择串口HMI显示屏 TJC3224T024\_011X，所选的显示屏如图5-7所示。



图 5-7 TJC3224T024\_011X显示屏

选择串口HMI显示屏的主要原因如下：

* 屏幕接受纯字符串指令，方便阅读与调试。
* 不占用主处理器存储。
* 布局方便。

### 5.4.2 显示屏资源

下表5-1是TJC3224T024\_011X显示屏的接口性能参数。

表 5-1 TJC3224T024\_011X显示屏的接口性能参数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | 测试条件 | 最小值 | 典型值 | 最大值 | 单位 |
| 串口波特率 | 标准 | 2400 | 9600 | 115200 | bps |
| 串口输出电平（TXD） | 输出 1，Iout=1mA | 3 | 3.2 | - | V |
| 输出 0，Iout=-1mA | - | 0.1 | 0.2 | V |
| 串口输入电平（RXD） | 输出 1，Iout=1mA | 2 | 3.3 | 5 | V |
| 输出 0，Iout=-1mA | -0.7 | 0 | 1.3 | V |
| 通讯模式 | 8,1,None | | | | |
| 用户接口方式 | 4Pin\_2.54mm 带锁扣 | | | | |
| SD 卡接口 | 有（FAT32 文件格式）最大支持 32G MicroTF/SD 卡 | | | | |

下表5-2是TJC3224T024\_011X显示屏的存储器特性。

表 5-2 TJC3224T024\_011X显示屏存储器特性

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 存储器类别 | 参数 | 最小值 | 典型值 | 最大值 | 单位 |
| FLASH 存储器 | 字库图片储存空间 | - | 4 |  | MB |
| 用户存储 | EEPROM |  | 无 |  | 无 |
| RAM 存储器 | 变量存储空间 | - | 3584 | - | BYTE |

下表5-3和下图5-8是外形参数。

表 5-3 TJC3224T024\_011X显示屏外形参数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **参数** |  |  |  | **数据** | |  |  |  |  |  | **说明** | |  |
|  |  |  |  | |  |  |  |  |  | |  |  |  |
| 颜色 |  | 65K（65536）色 | | | |  |  | 16 bit 调色板 5R6G5B | | | | |  |
|  |  |  |  | |  | | |  |  | |  |  |  |
| 外形尺寸 |  | 74.4（L）×42.72（W）×4.6（H） | | | | | | TJC3224T024\_011N | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 74.4（L）×42.72（W）×5.8（H） | | | | | | TJC3224T024\_011R | | | | |  |
|  |  |  |
|  |  |  |  | |  | |  |  |  | |  |  |  |
| 显示屏尺寸 |  | 60.26mm（L）×42.72mm（W） | | | | |  | 320×240 像素模式 | | | | |  |
|  |  |  |  | |  | |  |  |  | |  |  |  |
| 可视尺寸（A.A.） | | 48.96mm（L）×36.72mm（W） | | | | |  | 320×240 像素模式 | | | | |  |
|  |  |  |  | |  |  |  |  |  | |  |  |  |
| 分辨率 |  | 320×240 像素 | | | |  |  | 可设置成 240×320 | | | | |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |
| 触控类型 |  | 4 线精密电阻式 | | | |  |  |  |  |  |  |  |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |
| 触控次数 |  | 最少 100 万次 | | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 背光模式 |  | LED | |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |
| 背光寿命（平均） | | >30,000 小时 | | | |  |  |  |  |  |  |  |  |
|  |  |  |  | |  | |  |  |  | |  |  |  |
| 亮度 |  | 200nit（TJC3224T024\_011N） | | | | |  | 可进行 100 级亮度调节 | | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 180 nit（TJC3224T024\_011R） | | | | |  | 可进行 100 级亮度调节 | | | | |  |
|  |  |  |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  |
| 净重量 |  | 20g（TJC3224T024\_011N） | | | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 25.8g（TJC3224T024\_011R） | | | | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |

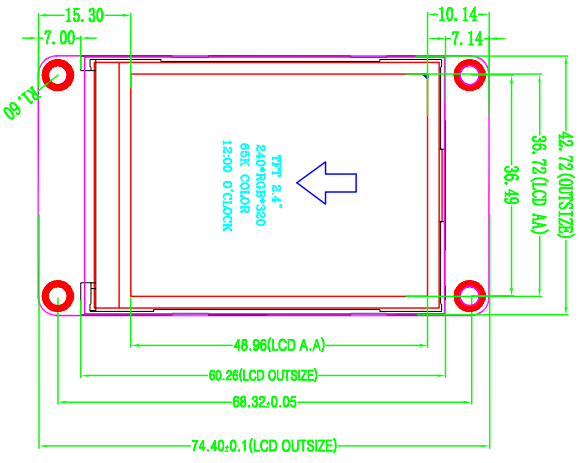


图 5-8 TJC3224T024\_011X显示屏外形参数

### 5.4.3 显示框架设计

显示模块是一个多状态的模块，所以选择状态机结构来实现显示功能。这个状态机的状态变量类型如下：

/\*

\* tft\_state - 在tft操作中，表示当前状态机的状态的类型

\* pgn表示页面编号，etn、ln分别一个页面中的元素号、行号，

\* objn表示元素号。

\*/

typedef struct tft\_state\_

{

uint8\_t pgn;

uint8\_t etn;

uint8\_t ln;

uint8\_t objn;

} tft\_state;

状态机结构的伪代码如下：

switch (tft\_stt.pgn)

{

switch (kvp\_menu[etn].attr)

{

}

}

关于显示的所有，最终都会体现在这个框架里，也就是所谓的状态机里，改变显示状态的原因是外部按键和旋钮的输入，逻辑简单，细节繁琐。

### 5.4.4 页面

页面指可以同时显示在显示屏上的对象的集合以及它们之间的布局关系。好的页面设计，可以方便同用户的交互过程。本设计的页面共有三个，分别如下所述。

第一个页面是开机起始界面，主要的目的是显示一个设计信息。如下图5-9所示。



图 5-9 起始界面

第二个页面是系统时间设置以及对象选择，对象选择即植物的选择，用于多植物管理。页面内容如下图5-10所示。



图 5-11 对象选择页面

第三个页面是对象属性的设置页面，对于每一个对象都有一个独立的设置页面，主要完成不同植物的计划数据的设置，页面内容如下图5-12所示。

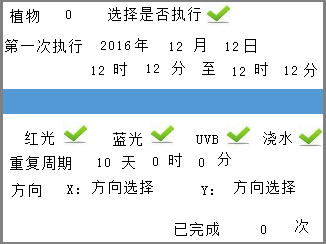


图 5-12 设置界面

## 5.5 时间模块

5.5.1 系统时间

系统时间在本设计中进行计划控制的一个基准值，所有的控制过程，都是在时间线上展开的。所以需要为系统提供一个稳定的时间模块。在时间模块中，主要提供的功能有：

* 设置系统时间。
* 读取当前时间。
* 不同时间格式的转换。

### 5.5.2 DS1302介绍

#### a. 概述

在时间模块中，选择DS1302芯片来实现系统时间。DS1302是美国DALLAS公司推出的一种高性能、低功耗的实时时钟芯片，其特点有：

* 实时时钟可以计数秒，分钟，小时，月，月日期，本周日和年份，闰年补偿，有效期至2100年。
* 31×8bit可以电池备份的通用随机存取存储器。
* 采用简单的SPI三线接口与MCu进行同步通信。
* TTL兼容。
* 在读写时钟信息或者RAM数据时，可以选择单字节传输或者选择突发模式下多字节传输。
* 低功耗操作使备份电池工作更长时间。
* 2.0V到5.5V电压范围内可以正常工作。
* 8管脚DIP和8管脚SO让空间需求最小化。
* 可选的工业温度范围在-40摄氏度到+85摄氏度。
* 安全监测实验室(UL)认证。

其外部封装如下图5-13所示。

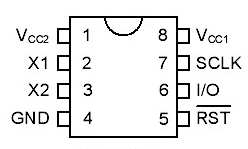


图 5-13 DS1302 8-Pin(300mil)

下图5-14展示了此计时器的主要元素：移位寄存器，控制逻辑，振荡器，实时时钟，以及RAM。



图 5-14 DS1302主要结构图

各引脚的功能为：

* Vcc1，主电源；Vcc2：备份电源。当Vcc2>Vcc1+0.2V时，由Vcc2向DS1302供电，当Vcc2< Vcc1时，由Vcc1向DS1302供电。
* SCLK，串行时钟，输入，控制数据的输入与输出；
* I/O，三线接口时的双向数据线；
* CE，输入信号，在读、写数据期间，必须为高。该引脚有两个功能：第一，CE开始控制字访问移位寄存器的控制逻辑；其次CE提供结束单字节或多字节数据传输的方法。

#### b．寄存器

DS1302有几组不同用途的寄存器，如下所述。

 有关日历、时间的寄存器共有12个，其中有7个寄存器（读时81h～8Dh，写时80h～8Ch），存放的数据格式为BCD码形式，如图5-15所示。

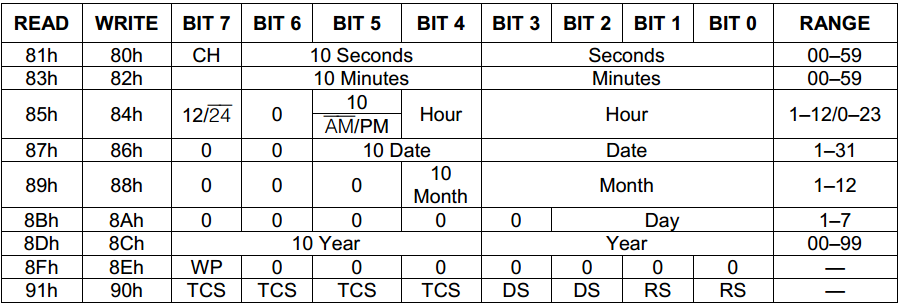


图 5-15 DS1302时间寄存器

小时寄存器（85h、84h）的位7用于定义DS1302是运行于12小时模式还是24小时模式。当为高时，选择12小时模式。在12小时模式时，位5是 ，当为1时，表示PM。在24小时模式时，位5是第二个10小时位。

    秒寄存器（81h、80h）的位7定义为时钟暂停标志（CH）。当该位置为1时，时钟振荡器停止，DS1302处于低功耗状态；当该位置为0时，时钟开始运行。

控制寄存器（8Fh、8Eh）的位7是写保护位（WP），其它7位均置为0。在任何的对时钟和RAM的写操作之前，WP位必须为0。当WP位为1时，写保护位防止对任一寄存器的写操作。

DS1302中附加31字节静态RAM的地址如图5-16所示。

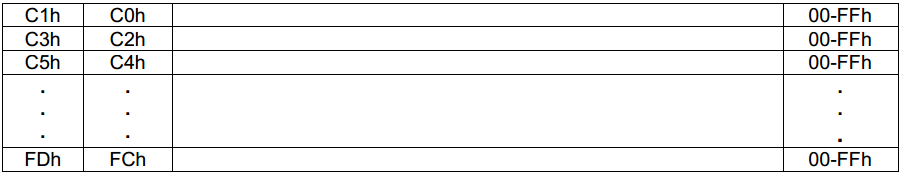


图 5-16 DS1302RAM寄存器

DS1302具有工作模式寄存器，可以改变此寄存器进入突发模式，突发模式也分为 RAM 突发模式和时钟突发模式，通过突发模式可以避免小概率的时间读取错误，寄存器如下图5-17所示。



图 5-17 模式寄存器

此外，DS1302还有充电寄存器等。

#### c. 读写时序

DS1302是SPI总线驱动方式。它不仅要向寄存器写入控制字，还需要读取相应寄存器的数据。要想与DS1302通信，首先要先了解DS1302的控制字。DS1302的控制字如图5-18所示。

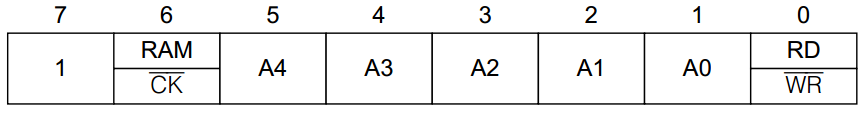


图 5-18 DS1302控制字

控制字的最高有效位（位7）必须是逻辑1，如果它为0，则不能把数据写入到DS1302中；位6如果为0，则表示存取日历时钟数据，为1表示存取RAM数据； 位5至位1（A4～A0）：指示操作单元的地址； 位0（最低有效位）：如为0，表示要进行写操作，为1表示进行读操作。

控制字总是从最低位开始输出。在控制字指令输入后的下一个SCLK时钟的上升沿时，数据被写入DS1302，数据输入从最低位（0位）开始。同样，在紧跟8位的控制字指令后的下一个SCLK脉冲的下降沿，读出DS1302的数据，读出的数据也是从最低位到最高位。数据读写时序如图5-19所示 。

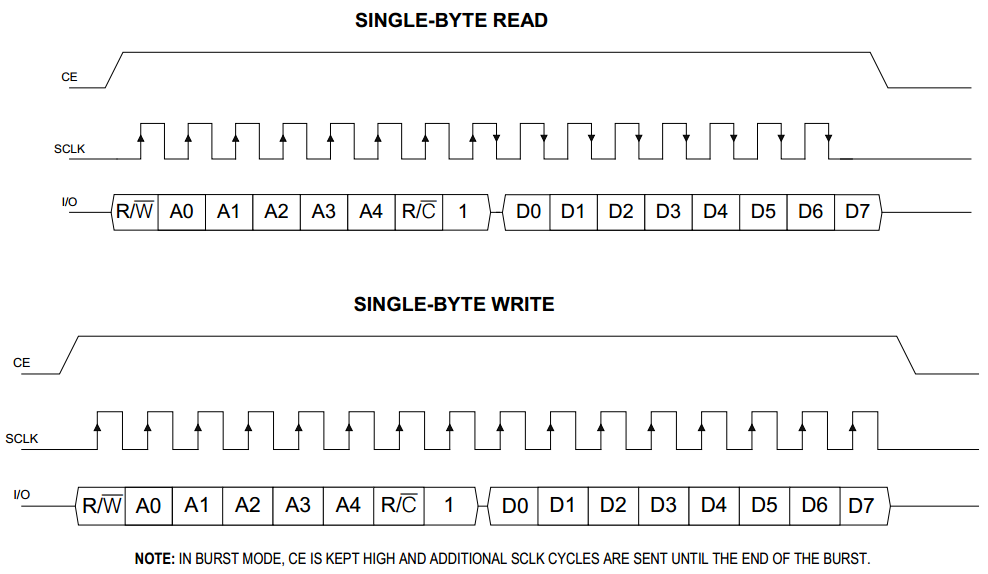


图 5-19 DS1302数据读写时序

### 5.5.3 时间接口

关于DS1302芯片的使用，最主要的是字节读写函数的编写，而在字节读写函数编写过程中，最主要的问题是时序要求。关于延时的实现，有指令循环延时和硬件定时器延时两种方法，指令循环延时不需要占用硬件定时器资源，但是不能做到足够的精确度，并且会占用CPU，硬件定时器延时可以做到比较高的精确度，并且不需要CPU的干预，但缺点是会占用额外的硬件定时器资源。在本设计中，由于DS1302的时序通过软件延时完全可以满足精度要求，并且总的延时时间很短并不影响其他任务，所以在时序中的延时部分使用的是软件延时，通过计算每一条指令的执行时间，并进行循环，来估算延时的总时间。基于字节的读写函数，可以很容易的写出时间模块对外的操作接口，如下：

void ds1302\_set\_time(calendar\_info \*cal);

void ds1302\_read\_time(calendar\_info \*cal);

void maintain\_system\_time(void);

calendar\_info get\_system\_time(void);

uint32\_t calendar\_to\_sec(calendar\_info \*cal);

calendar\_info sec\_to\_calendar(uint32\_t sec);

ds1302\_set\_time函数是用用户输入的时间设置DS1302的内部时间；ds1302\_read\_time函数是将DS1302内部的时间读取到内存中，maintain\_system\_time函数可以通过定时调用以在内存中维护一个存储有当前时间信息的数据区域；get\_system\_time函数得到在内存中维护的当前时间；calendar\_to\_sec函数是分解时间到日历时间的转换，sec\_to\_calendar函数是日历时间到分解时间的转换，这两个函数的实现是为了方便时间值的运算。

## 5.6 数据保存

### 5.6.1 数据保存介质

用户可以通过本装置来输入数据，并且在运行时也会产生一些有意义的数据，本设计的一个目标是持久的保持这些有效数据。当装置上电时，这些数据被保存在RAM中，但是当装置被断电时，这些在RAM中的数据将要丢失，所以需要在断电时可以保存数据的存储介质，在断电时将RAM中的数据保存到此介质中，在上电时重新将数据从此介质中恢复到RAM中。可选的存储介质有EEPROM、FLASH，以下将详细对EEPROM、FLASH这几种存储介质进行比较。

EEPROM、EPROM、FLASH 都是基于一种浮栅管单元(Floating gate transister)的结构。EPROM的浮栅处于绝缘的二氧化硅层中，充入的电子只能用紫外线的能量来激出，EEPROM的单元是由FLOTOX(Floating- gate tuneling oxide transister)及一个附加的Transister组成，由于FLOTOX的特性及两管结构，所以可以单元读/写。技术上，FLASH是结合EPROM和EEPROM技术达到的，很多FLASH使用雪崩热电子注入方式来编程，擦除和EEPROM一样用 Fowler-Nordheim tuneling。但主要的不同是，FLASH对芯片提供大块或整块的擦除，这就降低了设计的复杂性，它可以不要 EEPROM单元里那个多余的Tansister，所以可以做到高集成度，大容量，另FLASH的浮栅工艺上也不同，写入速度更快。

对于用户来说，EEPROM和FLASH 的最主要的区别是：

* EEPROM 可以按位擦写，而FLASH 只能按扇区擦除。
* EEPROM相对于flash来说，一般用于小容量、低端的产品。
* EEPROM的可重复写擦除入的次数大于flash。
* FLASH成本较EEPROM高，在集成度、一次写入速度上也较好。

结合获得的便利性以及系统需求，由于系统的MCU的片内外设包括flash，所以选择flash为断电保存数据的存储介质。

### 5.6.2 flash介绍

所选的处理器内部的flash特点如下：

* 扇区大小为 1KB。
* Flash 存储器保护方案可以阻止已存储数据的意外写入或者擦除。
* 自动的，内置的， 带验证的写入和擦除算法。
* 内部高电压源为 Flash 存储器的写入和擦除操作提供了动力。
* 可选的中断发生器取决于 Flash 命令完成。
* MCU 安全机制可以阻止未授权访问 Flash 存储器内容。

模块的框图如下图5-20所示。

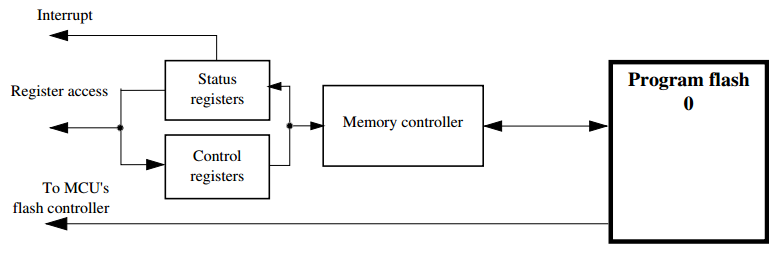


图 5-20 FLASH模块框图

### 5.6.3 flash操作驱动

使用flash的目的是在需要时写入或者读取数据，所以，设计的操作接口如下：

void flash\_write(uint8\_t \*saddr, uint16\_t nbyte);

void flash\_read(uint8\_t \*daddr, uint16\_t nbyte);

其中，函数flash\_wirte函数用于向flash中写入需要被保存的数据，函数flash\_read用于从flash中读取保存的数据。下面描述flash驱动实现的一些细节。

* 由于程序代码和被保存的数据都在flash中，所以需要注意这两个部分之间不能有地址重合，在iar生成的map文件中可以查看目标文件对存储器使用情况。
* 由于flash在写入的时候不能cpu不能从flash中读取指令，所以，对flash的写入函数需要重定位到RAM中，在本设计中重定位的方法为，首先在RAM中申请一块足够大的静态存储空间，然后在flash初始化的时候，将flash中的flash写入函数复制到之前申请到的空间中，将此空间的地址转换为flash写入函数的地址以在写入时调用。
* 写入数据的字节数必须以4对齐。
* 每次在写入的时候，需要先擦除flash中对应的空间，将所有的擦除为1，所选的flash的扇区大小是1KB，所以一次擦除的最小大小为1KB，擦除之后的空间可以被重新写入。
* 对于flash的操作需要确定flash的配置参数，包括块基地址、快大小等，如下内容：

FLASH\_SSD\_CONFIG flashSSDConfig =

{

FTFx\_REG\_BASE, /\* FTFx control register base \*/

PFLASH\_BLOCK\_BASE, /\* base address of PFlash block \*/

PBLOCK\_SIZE, /\* size of PFlash block \*/

DEFLASH\_BLOCK\_BASE, /\* base address of DFlash block \*/

0x0U, /\* size of DFlash block \*/

EERAM\_BLOCK\_BASE, /\* base address of EERAM block \*/

0x0U, /\* size of EEE block \*/

DEBUGENABLE, /\* background debug mode enable bit \*/

NULL\_CALLBACK /\* pointer to callback function \*/

};

## 5.7 控制输出

### 5.7.1 方向控制

在多植物管理的时候，需要调整装置的执行部分的方向来以管理不同的植物。装置的执行装置有不同光谱的灯以对植物进行补光，在切换补光对象时，需要旋转方向使补光方向改变，所以本设计具有方向控制的功能。

在方向控制的硬件实现上，使用的是两个一维云台来实现二维方向的控制。选用的云台各技术参数如下图5-21所示。



图 5-21 云台计术参数

在方向控制部分，对外提供的主要接口函数有如下几个：

void orient\_setspeed(uint8\_t addr, orient ori, uint8\_t spd);

void orient\_presetop(uint8\_t addr, preset\_op op, uint8\_t npre);

void orient\_setmode(uint8\_t addr, orient\_mode mod);

函数orient\_setspeed用于设置旋转的方向以及速度，函数orient\_presetop用于云台的预置点操作，函数orient\_setmode用于设置云台工作模式。

### 5.7.2 灯光控制

本装置能够按计划对植物进行光照的补偿，因此具有控制灯光的功能，灯光的控制是一个开关量的控制，选择使用继电器来实现这一控制功能。所选的继电器为松乐SRD-05VDC-SL-C，其参数在下图5-22中说明。

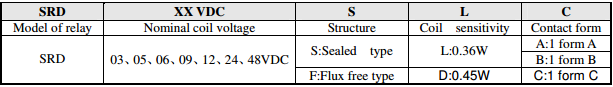


图 5.22 松乐继电器型号参数

其结构如下图5-23所示。

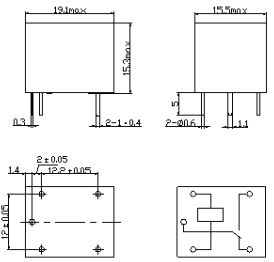


图5-23 继电器结构

继电器的是直接通过处理器的io电平来控制的。其相关的数据结构有：

typedef enum lg\_state\_

{

LIGHT = 1, NO\_LIGHT = 0

} lg\_state;

lg\_state类型表示灯光的开关，LIGHT为开灯，NO\_LIGHT为关灯。

## 5.8 软件开发环境

在软件开发中，使用的是IAR Systems公司的Embedded-Workbench for MSP430开发平台。

IAR Systems是全球领先的嵌入式系统开发工具和服务的供应商。公司成立于1983年，迄今已有27年，提供的产品和服务涉及到嵌入式系统的设计、开发和测试的每一个阶段，包括：带有C/C++编译器和调试器的集成开发环境(IDE)、实时操作系统和中间件、开发套件、硬件仿真器以及状态机建模工具。

国内普及的MSP430开发软件种内不多，主要有IAR公司Embedded Workbench for MSP430（简称为EW430）和AQ430。 目前IAR的用户居多。IAR EW430软件提供了工程管理，程序编辑，代码下载，调试等所有功能。

EW430的主要界面如下图5-24所示。

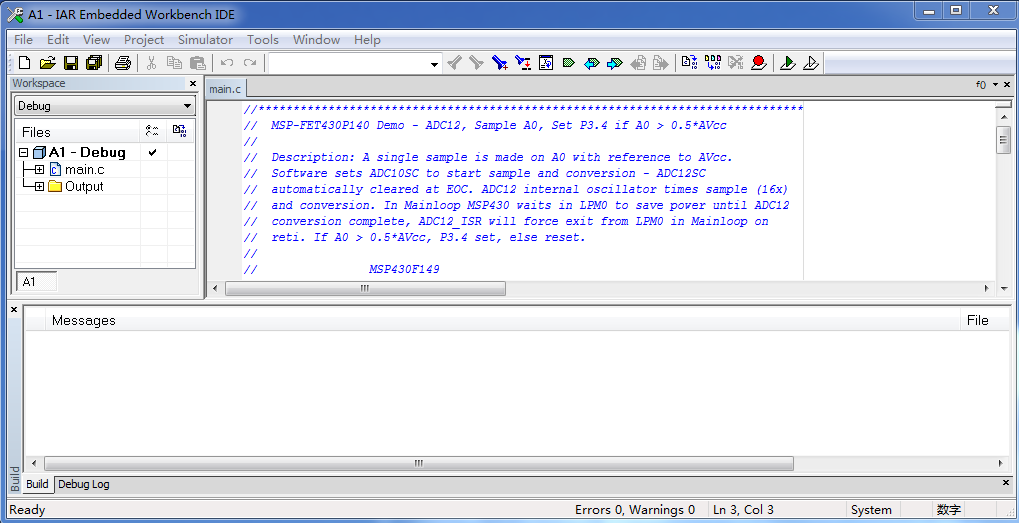


图 5-24 EW430界面

在调试中，使用的调试工具是MSP-FET，MSP-FET 是一款强大的仿真开发工具（通常称为调试探针），可帮助用户在 MSP 低功耗微控制器 (MCU) 上快速开始应用开发。创建 MCU 软件通常需要将生成的二进制程序下载到 MSP 器件中，以进行验证和调试。MSP-FET 在主机和目标 MSP 之间提供调试通信通道。此外，MSP-FET 还在计算机的 USB 接口和 MSP UART 之间提供反向通道 UART 连接。这就为 MSP 编程器提供了一种在 MSP 和运行在计算机上的终端之间进行串行通信的便利方法。它还支持使用 BSL（引导加载程序）通过 UART 和 I2C 通信协议将程序（通常称为固件）加载到 MSP 目标中。USB 接口将 MSP-FET 连接至计算机，而 14 引脚连接器提供对 MSP 调试仿真端口的访问，该端口包含标准 JTAG 接口或使用省引脚 Spy-Bi-Wire（2 线式 JTAG）协议。MSP-FET调试器硬件如下图5-25所示。



图 5-25 MSP-FET调试器

在iar中FET仿真器的配置界面如下图5-26所示。

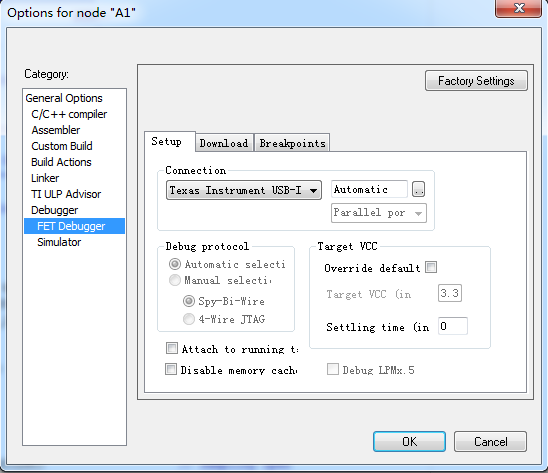


图 5-26 iar中FET仿真器的配置界面

# 6 运行结果

## 6.1 装置部件

在完成了本设计的硬件软件设计、实现以及调试之后，最终实现的植物管理辅助装置已经可以满足在需求分析时定下的要求，装置的各部件介绍如下：

操作面板部分，操作面板是装置实现人机交互的主要部分，如下图6-1所示。



图6-1 操作面板

光照部分，主要实现补光的执行，如图6-2所示。



图 6-2 光照部分

方向控制部分，是用云台实现的，主要用于多植物管理，如下图6-3所示。



图 6-3 方向控制

## 6.2 自动模式

装置可以运行于自动模式下，通过输入的计划数据自动的管理植物，相关部分如下所述。

选择植物步骤，如下图6-4所示。



图 6-4 选择植物

设置选定植物计划数据，如下图6-5所示。



图 6-5 设定计划数据

## 6.3 手动模式

装置可以运行于手动模式下，此时，执行装置有用户直接控制，如下图6-6所示。



图 6-6 手动控制

# 7 毕业设计总结

本次毕业设计从需求分析经过设计、执行到最终完成，经过了一个不算短的时间。实现了预期的目标，可以正常完成辅助管理植物的功能。比较重要的问题总结如下：

关于本设计需要实现的功能，将会影响到整个设计的实现，所以，应该在最开始的时候尽量详细完善功能设计，如果要在后期更改功能要求的话，将会影响之前所做的工作，造成进度的延迟。

关于整个软件系统框架的设计，是需要经过深思熟虑的，一个好的框架，会给整个系统带来本质上的提升，在扩展功能、提升效率、降低成本、加快进度上有明显的改变，在本次设计中，对每一个硬件模块做了模块划分，使系统的实现不必担心硬件模块的细节，但是没有在逻辑层次上再次进行模块划分，导致逻辑不是很清晰，在扩展功能、更改细节问题上不能很方便的实现，这是一个有待改进的问题。

# 致谢

在本次毕业设计的完成过程中，从设计到实现以及论文的完成，离不开周围老师同学的帮助。首先感谢的是我的导师亢红波老师，在整个设计期间，亢老师给我提供了完成设计的方向，解决了许多困扰我的问题。在这里衷心地感谢亢老师不辞辛苦的指导和帮助。

同时感谢我周围的同学，感谢孙大敬同学在本次设计实现中对硬件部分提供的支持，使我能够顺利的完成整个装置的实现。并且感谢自动化217、236实验室在我毕设过程中提供的帮助，感谢实验室同学的理解以及实验室提供对我提供的资源。

最后感谢度过我四年大学时光的母校—西安邮电大学，在四年大学时光里，我认识了值得珍惜的朋友，也学到了珍贵的知识，这里是我应该永远记住的回忆。

# 参考文献

# 附录

## 附录1 部分程序代码

/\*

\* key\_read\_base - 按键基础读取函数，

\* @key\_info: 要检测按键的所需信息,必须在是全局变量区中存储的

\*

\* 这个函数返回 N\_KEY,S\_KEY,L\_KEY三种情况

\*/

key\_mean key\_read\_base(volatile key\_infomation \*key\_info)

{

state\_sign key\_state\_io;

key\_mean key\_m = N\_KEY;

// 获取io状

key\_state\_io = (\*((\*key\_info).get\_state\_io))();态

switch((\*key\_info).state\_base)

{

case key\_state\_0:

if (key\_state\_io == VALID)

{

// 按键按下后首先进入消抖和按键确认状态

(\*key\_info).state\_base = key\_state\_1;

}

break;

case key\_state\_1:

if (key\_state\_io == VALID)

{

// 按键如果仍然按下，则消抖完成,状态转换,此时返回无按键事件

(\*key\_info).time\_base = 0;

(\*key\_info).state\_base = key\_state\_2;

//

}

else

{

(\*key\_info).state\_base = key\_state\_0;

}

break;

case key\_state\_2:

if (key\_state\_io == INVALID)

{

// 此时按键释放说明为一次短操作

key\_m = S\_KEY;

(\*key\_info).state\_base = key\_state\_0;

}

// 继续按下，计时加10ms，即本函数调用周期

else if (++(\*key\_info).time\_base > 100)

{

// 按下时间大于1s，返回长按键,转换为等待按键释放状态

key\_m = L\_KEY;

(\*key\_info).state\_base = key\_state\_3;

}

break;

case key\_state\_3:

if (key\_state\_io == INVALID)

{

// 按键此时已释放，转换为初始状态

(\*key\_info).state\_base = key\_state\_0;

}

break;

default:

break;

}

return key\_m;

}

/\*

\* key\_read - 按键读取函数

\* @key\_info: 要检测的按键所需的信息，必须是在全局变量区中存储的

\*

\* 这个函数会返回N\_KEY,S\_KEY,D\_KEY，L\_KEY四情况

\*/

key\_mean key\_read(volatile key\_infomation \*key\_info)

{

key\_mean key\_m = N\_KEY,

key\_m\_temp;

key\_m\_temp = key\_read\_base(key\_info);

switch((\*key\_info).state)

{

case key\_state\_0:

if (key\_m\_temp == S\_KEY)

{

// 第一次单击状态，仍然返回无键，

// 到下个周期判断是否出现双击

(\*key\_info).time = 0;

(\*key\_info).state = key\_state\_1;

}

else

{

// 对于非单击，返回原事件

key\_m = key\_m\_temp;

}

break;

case key\_state\_1:

if (key\_m\_temp == S\_KEY)

// 又一次单击，间隔 < 300ms

{

key\_m = D\_KEY;

(\*key\_info).state = key\_state\_0;

}

else

{

if (++(\*key\_info).time > 30)

{

// 300ms内没有再出现单击事件，则返回上一次单击事件

key\_m = S\_KEY;

(\*key\_info).state = key\_state\_0;

}

}

break;

default:

break;

}

return key\_m;

}

/\*

\* tft\_page\_refresh() - 在页面信息有改变时调用以刷新页面

\*/

void tft\_page\_refresh(void)

{

switch (tft\_stt.pgn)

{

case ORIGINAL\_PG:

for (uint8\_t etn = 0; etn < sizeof(kvp\_original) / sizeof(kv\_pair);

etn++)

{

switch (SW\_PAGE)

{

case R\_NUM:

break;

case RW\_NUM:

break;

case RW\_PIC:

break;

case SW\_PAGE:

break;

default:

break;

}

}

break;

case MENU\_PG:

for (int etn = 0; etn < sizeof(kvp\_menu) / sizeof(kv\_pair);

etn++)

{

switch (kvp\_menu[etn].attr)

{

case R\_NUM:

case RW\_NUM:

sprintf(tft\_cmd\_str, "%s.val=%d", kvp\_menu[etn].key,

kvp\_menu[etn].value);

tft\_send\_cmd(tft\_cmd\_str);

sprintf(tft\_cmd\_str, "ref %s", kvp\_menu[etn].key);

tft\_send\_cmd(tft\_cmd\_str);

break;

case RW\_PIC:

sprintf(tft\_cmd\_str, "vis %s,%d", kvp\_menu[etn].key,

kvp\_menu[etn].value);

tft\_send\_cmd(tft\_cmd\_str);

break;

case SW\_PAGE:

break;

default:

break;

}

}

break;

case OBJ\_SET\_PG:

for (int etn = 0; etn < sizeof(kvp\_obj\_set[0]) / sizeof(kv\_pair);

etn++)

{

switch (kvp\_obj\_set[tft\_stt.objn][etn].attr)

{

case R\_NUM:

case RW\_NUM:

sprintf(tft\_cmd\_str, "%s.val=%d",

kvp\_obj\_set[tft\_stt.objn][etn].key,

kvp\_obj\_set[tft\_stt.objn][etn].value);

tft\_send\_cmd(tft\_cmd\_str);

sprintf(tft\_cmd\_str, "ref %s",

kvp\_obj\_set[tft\_stt.objn][etn].key);

tft\_send\_cmd(tft\_cmd\_str);

break;

case RW\_PIC:

sprintf(tft\_cmd\_str, "vis %s,%d",

kvp\_obj\_set[tft\_stt.objn][etn].key,

kvp\_obj\_set[tft\_stt.objn][etn].value);

tft\_send\_cmd(tft\_cmd\_str);

break;

case SW\_PAGE:

break;

default:

break;

}

}

break;

default:

break;

}

return;

}