Very Kerbal Controller

Design Document

Table of Contents

[System Overview 3](#_Toc458204319)

[Physical Design 4](#_Toc458204320)

[Electrical Design 5](#_Toc458204321)

[General Design Concepts 5](#_Toc458204322)

[Power Supply Design 5](#_Toc458204323)

[Potential Changes 5](#_Toc458204324)

[Input Sensing 5](#_Toc458204325)

[Potential Changes 6](#_Toc458204326)

[Driving Outputs 6](#_Toc458204327)

[Software Design 7](#_Toc458204328)

# System Overview

# Electrical Design

## General Design Concepts

The electrical design was based on the following basic concepts:

* Maximise use of parts on hand, or choose minimum cost approaches
* All inputs to use Pull-Up resistors so that all inputs are consistent and the internal Pull-Ups in the Arduino can be used
* Connectivity setup to minimise wiring runs. Panel segments grouped to nearby components.

## Power Supply Design

The power supply was driven by the highest required voltage which is 12VDC required by the LED strips. The two step down voltages were then chosen as follows:

* 5VDC was required for the fan as this was an existing part and buying a 12VDC fan was unnecessary cost. A fan is $5-$10 whereas a 7805 regulator is $1.
* 8VDC was not strictly required as the Arduino will accept 12VDC in the Japan Jack. However some online reading of temperature testing indicated the on-board voltage regulator can get quite warm at 12VDC. The 7808 regulator therefore provides an intermediate drop to spread the heat. 8VDC was chosen only because it’s the next 78XX regulator above the Arduino’s minimum 7.25VDC input.

The regulators are bolted directly to the rear plate for heat dissipation, but given the low current this is not really required.

The fan is a 3 wire type however the third wire (tach) cannot be used when the input power is PWM switched as you are also switch power to the sensor. Therefore it is not connected. Also testing showed the tach signal didn’t work anyway.

Required current was based on 54 LEDs at 20mA which is approximately 1A, plus maximum of 500mA for the Arduino. Allowing for other loads like the fan a 2.5A supply allows room for growth.

### Potential Changes

Replace the fan with a 12VDC fan and remove the 5VDC conversion. Also either use a 2 wire fan or, if speed sensing is required, a 4 wire fan. For a 4 wire fan the PWM transistor can be removed and the PWM signal connects directly to the 4th wire.

## Input Sensing

All digital inputs are grounds via Pull-Up resistors, either internal for direct connections to the Arduino or external resistors for the extenders. This results in sensing 5V, or HIGH, when the switch is off but this can be reversed in the code to give a more readable output.

All switches used N -1 inputs where N is the number of switch position. E.G. a 12 position switch uses 11 inputs and if all 11 inputs are off it must be at position 12.

All analogue inputs, except the voltage sensor, are powered from the Arduino 5VDC supply.

The voltage sensor at A0 is to allow the code to sense the source of input power because the Vin pin will show the current input voltage. If the Japan Jack is powered the USB power supply is cut off and Vin is equal to the Japan Jack voltage, otherwise it is the USB voltage.

However this means that if Vin is connected to A0 and the Japan Jack is > 5VDC then the Arduino will be damaged. Therefore a simple voltage divider scales the sensed voltage to ensure it is always less than 5VDC, in this case A0 will be 1/3 of the input voltage.

### Potential Changes

The interrupt pins on the MCP23017 extenders are currently not used. If they were connected to digital inputs on the Arduino they would allow the extender to flag a change on an input to the Arduino so the Arduino would not need to poll the inputs all the time.

## Driving Outputs

To drive the 12VDC lights the ULN2803 Darlington arrays are used, though as there are 9 lights per side a 9th switching channel was created with a simple NPN transistor. A diode from this resistors collector the test switch replicates the common connection in the Darlington array. A base resistor, and a pull down resistor is required for this transistor whereas in the Darlington Array they are built in.

The ground of the driver arrays is PWM dimmed using another NPN however this creates a potential issue. This PWM transistor also requires a base resistor and a pull down resistor.

When only some lights are active, there is a path to ground for those lights back through the LOW signals switch off the other lights, as outputs are low impedance. This is prevented by adding diodes between the extender and the Darlington array forcing the PWM to ground path to be used.

An alternative would be to use High Side switch and place the dimming transistor on the power side of the LEDs but I am not confident with this approach and it would require 12V to go via the circuit board rather than direct to the lights.

## Grounding

A hybrid star grounding approach is used throughout. Each panel has a star ground bar that joins the grounds for all elements on that panel. The only exception is the I2C extender board on the back of the main panel which is grounded back to the Arduino.

From the main power ground the following ground points are established

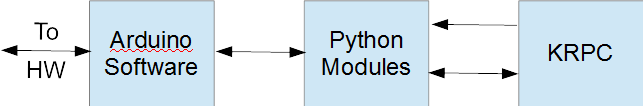
* Arduino Japan Jack
* Side Panel ground Bar
* Main Panel Ground bar
* 7805 and 7808 voltage regulator ground.

**Note the ground bars on the panels are NOT electrically joined to the panel. They are held with hot glue that does not proved connectivity and as such the metal panels are NOT grounded. However given the low voltage supply in use this is considered low risk. All panels and metal case components should normally be grounded to provide an alternate path than your body to ground!**

# Software Design

## Software Architecture

The overall SW Architecture is as follows:



The Arduino manages all of the communication with the hardware. It then communicates via Serial over USB to the Python Models as per the interface defined below. The Python modules communicate directly to KRPC as documented by KRPC. Note there are two connections from the python modules to KRPC, one to read data for the GUI and another two way connection for panel interfacing. Refer to the Python Code section for more details.

### Arduino – Hardware Interface

Refer to the schematics for the interface from the physical HW to the Arduino.

### Arduino - Python Interface

The Arduino transmits a series of bytes to the Python Module when requested. The message structure is shown in the table below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Byte** | **Value** | **Description** | **Byte** | **Value** | **Description** |
| 0 | Status | b0 = alive, b1 = powered | 13 | Analogue 0 | Voltage Sense |
| 1 | Digital Byte 0 | Pins 8-10 | 14 | Analogue 1 | Temperature Sense |
| 2 | Digital Byte 1 | Pins 22-29 | 15 | Analogue 2 | Dimmer Setting |
| 3 | Digital Byte 2 | Pins 30-37 | 16 | Analogue 3 | Rotation X |
| 4 | Digital Byte 3 | Pins 38-45 | 17 | Analogue 4 | Rotation Y |
| 5 | Digital Byte 4 | Pins 46-53 | 18 | Analogue 5 | Rotation Z |
| 6 | Digital Byte 5 | Mux 0 Bank A | 19 | Analogue 6 | Translation X |
| 7 | Digital Byte 6 | Mux 0 Bank B | 20 | Analogue 7 | Translation Y |
| 8 | Digital Byte 7 | Mux 1 Bank A | 21 | Analogue 8 | Translation Z |
| 9 | Digital Byte 8 | Mux 1 Bank B | 22 | Analogue 9 | Throttle |
| 10 | Digital Byte 9 | Mux 2 Bank B | 23 | Frame Time |  |
| 11 | Digital Byte 10 | Mux 3 Bank B | 24 | Fan Speed |  |
| 12 | Digital Byte 11 | Mux 4 Bank B |  |  |  |

The Python Model transmits a series of bytes to the Arduino as defined in the table below.

|  |  |  |
| --- | --- | --- |
| **Byte** | **Value** | **Description** |
| 0 | CMD | b0 = request input data, b1 = process outputs |
| 1 | D0 | Left Light Commands – Fuel, Monoprop, Fuel Low, Monoprop Low |
| 2 | D1 | Right Light Commands – Battery, Overheat, Battery Low, Temperature High, All Gear/Legs Up, All Gear/Legs Down |

Note that while there are 18 lights, some lights are driven directly from the sensed HW switch. This applies to all advisory lights except Gear. Gear is handled as a special case to allow ‘in transit’ light flashing.

## Arduino Code

### Arduino Environment

There are no customisations for the Arduino Environment required. The standard IDE was used.

### Arduino Code Structure

The Arduino code is divided into 3 files as noted below. This was done only for readability.

#### Main Module – KSP Controller.C

The main module follows the Arduino standard of ‘Setup’ and ‘Main’. Main calls out two other functions, ‘read\_inputs and ‘write\_outputs’ only to aid readability of the main module.

An outline of the code noting any special approaches is as follows

* Setup
  + Set the PWM frequency to max to avoid fan whine1
  + Set the output pin mode for output pins
  + Turn on all LEDs as a light check during setup
  + Activate internal pullup resistors on input pins
  + Force unused analogue inputs to digital low to prevent noise2
  + Activate the I2C bus and check that all extender chips respond
    - If not, raise a critical error – Code 1
  + Set output mode on the extender pins as required
  + Reverse polarity on the extender input pins as required3
  + Activate serial communication
  + After a short delay, switch off all LEDs to end the setup test
* Main
  + Control frame timing to 10ms and track time since last power on
  + If serial data is available, read it in and decode the command bit into instruction flags
    - Also track the time since the last serial message was received
  + Check the input power level, if it is USB only flash the error light and skip remaining processing
    - If serial has not been received for some time, clear all outputs and flash the error light, skip data processing
      * Read in hardware inputs 🡪 see ‘read\_inputs’ function
      * If requested, process outputs 🡪 see ‘write\_outputs’ function
      * Set the light dimmer output to the requested dimming level
    - Manage the fan speed4
    - Check temperature readings
      * If high, raise a critical error – Code 2
    - Measure the actual processing time of the frame
      * Add this to the send buffer
      * If it exceeds the limit flash the Overrun light
    - If requested to return input date, send the buffer
* ‘read\_inputs’ function
  + Read in the direct input pins and bit stuff them into bytes
  + Read in the input bytes from the extenders
  + Read in the analogue values and map them to 0-255 so they can be stored in a byte
    - Note where required this allows analogues to be reversed when they have been wired opposite to expectation
* ‘write\_outputs’ function
  + Combine received light commands with digital inputs from switches to form an output byte
  + Transmit the byte to the extender
  + Repeat for each affected bank
    - Note where only some pins are used as outputs any other data sent in the byte is ignored by the extender so no special handling is required
  + Note the special handling of the gear indication.
    - If all gear/legs up 🡪 light off
    - If all gear/legs down 🡪 light on
    - Otherwise light flashes

**Notes:**

1. Switching power to a fan at a frequency in an audible range (20-20kHz) produces a whine. The Arduino allows the PWM frequency to be modified to a range of values. Note that pins share a PWM timer as follows:
   * timer 0 controls pin 13, 4
   * timer 1 controls pin 12, 11
   * timer 2 controls pin 10, 9
   * timer 3 controls pin 5, 3, 2
   * timer 4 controls pin 8, 7, 6

Note that timer 0 also is used for timing functions, changing this frequency will change performance of timing functions like millis() etc.

The available frequencies and their associated timer value are:

* + 1 ---> PWM frequency is 31000 Hz
  + 2 ---> PWM frequency is 4000 Hz
  + 3 ---> PWM frequency is 490 Hz (default)
  + 4 ---> PWM frequency is 120 Hz
  + 5 ---> PWM frequency is 30 Hz
  + 6 ---> PWM frequency is <20 Hz

Note that timer 0 frequencies are double the values above

1. Unconnected analogue inputs are antennae and as they share a common connection with other analogue inputs will affect readings if left open and floating. This can be avoiding by setting the pinmode to output so they act as a digital output and forcing the value to digital low. This grounds the input and avoids noise.
2. Using pull up resistors gives 5v (HIGH) when the switch if off and 0v (LOW) when the switch is on. This can be difficult to read directly so reversing the polarity is useful. The MCP23017 provides this function internally so no processing is required. However in setting this function in the chip note the following constraints:
   * The Stage switch is Normally Closed, that is, pressing the switch breaks the contacts. This effectively reverses the polarity mechanically and so the extender should be set to not reverse polarity on this pin.
   * Pin banks that combine inputs and outputs should only set reverse polarity on inputs.
3. The fan cannot be speed controlled over the full range. The fan motor has a minimum speed at which it will start, and then a lower speed at which, if already running, the fan would stall. If the fan is stalled but a low speed is requested the power sent to the fan becomes heat which should be avoided.
   * To avoid this problem the fan is set to full speed for a fixed time after power up. After this the fan speed is a linear interpolation of temperature that gives a lowest fan speed well above the stalling speed at low temperature increasing to full speed a few degrees below the critical temperature where an error would be declared.

#### Variables and Constants – KSP Controller.h

This header module contains:

* Defined values
* Variables including initialisation values
* Constants
* Function prototypes for main module functions

Each constant, including units where applicable, are commented in the code.

#### Utilities – utils.h

This header module contains utility functions used throughout the main module. The functions are:

* sys\_error
  + This function is called when a critical error is encountered. The function flashes a repeating code (‘code’ input) on the error light as sent by the caller. The flashes are 200ms apart with a 1000ms gap between repeats to that the user can determine which error occurred.
  + The function is block via an endless while loop. An Arduino restart is required to clear a critical error
* light\_flash
  + This function will cause a given pin to flash, changing from on to off or vice-versa at the given interval
  + It DOES NOT check the pin given is an output.
* light\_hold
  + This function is a ‘delay off’
  + When the input condition is met the given pin is held high for the nominated hold time even if the input condition has become false.
  + It DOES NOT check the pin given is an output.
* mux\_Tx
  + This function transmits a byte of data to a given address and register on the I2C bus.
  + It is used to communicate to the MCP23017 extenders
* mux\_Rx
  + This function requests a number of bytes from a given address and register on the I2C bus.
  + The data is returned in the provided byte container

## Python Code