mk14keyscp

A MK14 keys interface using Charlieplexing by David Allday.

The PCB is designed to allow either an Arduino nano or a NodeMCU-ESP8266 to control the optical relays.

Using Charlieplexing works best if all the LEDs are of the same type so use the same type of optical relay in all sockets.

The 7 lines are also presented at the top of the board and could be attached to any device that can pull them high, low and has high impedance mode.

Optical Relays needed

You actually don’t need to populate all the optical relays as it was designed to cover all the cross overs.
When using the Issue V or Issue VI of the board then edge connector lines 6 and 8 are not used.
This means you could leave U13 and U11 sockets empty.
When using the JMP board it does not use edge connector lines 3 and 4, which means you could leave U15 and U16 sockets empty.
Please check the schematic if you are going to leave any sockets empty as I may have got it wrong.

MK14 reset switch

There are 2 optical relays are used to handle the reset switch.
U5 is used when you connect flying leads to the reset capacitor on the original MK14 boards.
U6 is used if using the newer replica Issue VI boards where the reset is present on the edge connector.
You can populate either U5 or U6, or both if you want to switch between board types.
This was simpler than trying to sort out jumpers to switch between the different reset options and my experiments showed it was capable of driving both optical relays at the same time.

MK14 OS select

There is a jumper attached to Arduino A7 and the ESP8266 A0 which can be used to tell the firmware which version of the MK14 OS it is talking to. Add jumper to use the original OS that starts with ---- --, leave open for the later versions.

Wifi status

One of the bits missing from the board is a way of telling the wifi status.
It can be either connected to the previously supplies SSID or it can have created it’s own Wireless Access Point (WAP mode).
There are 3 reasons why it go into WAP mode.

1. If there is no SSID stored in the system ROM.
2. It tries and fails to connect to the SSID stored in the system ROM.
3. The reset button is pressed twice within 5 seconds, called double reset.

   Note:- For double reset to work the reset button should be pressed twice with 2 seconds but not too quickly as the ESP8266 needs to at least get into the code before the second reset occurs. You can adjust the time period by changing the value defined by DRD_TIMEOUT in the code.

To give basic status information you need to wire 2 leds to the GPIO0 pin (D3).

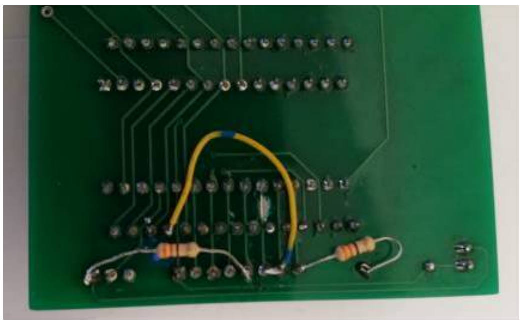
This is the pin that is held low to enable “flash” mode but it can be used for output with care. The additional circuit is,

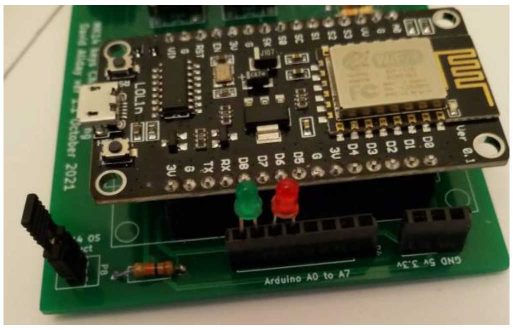
5v --- R330 --- GreenLED --- GPIO0 --- RedLED --- R330 --- 0v

see mk14schemaMod1.pdf

When GPIO0 is set as input then both leds should light, actually the red glows not very bright.
When GPIO0 is set as output high or low just one led lights.
I thought I’d need to be careful as if GPIO0 is set as output high and the programming button is pressed they may be a direct connection to Ground and that might cause a short :(
Luckily on the ESP8266 LOLIN v3 there seems to be a 200R resistor between GPIO0 and the programming button so all is well.

I found it quite easy to put a resistor from the gnd on the power outlet to the what would be A3 on the Arduino output socket, and from 5v side of the OS select resistor to the A0 on the Arduino output socket. Then connected A1 and A2 on the Arduino output socket to D3 (GPIO0) on the ESP8266. Then leds could be plugged into the Arduino output sockets.





Let me know if you find a better way of doing this.

I’m sure there is a number of improvements that could be made to the board and I might get around to making version 2 but there again . . . .

Stay safe
David Allday
November 2021