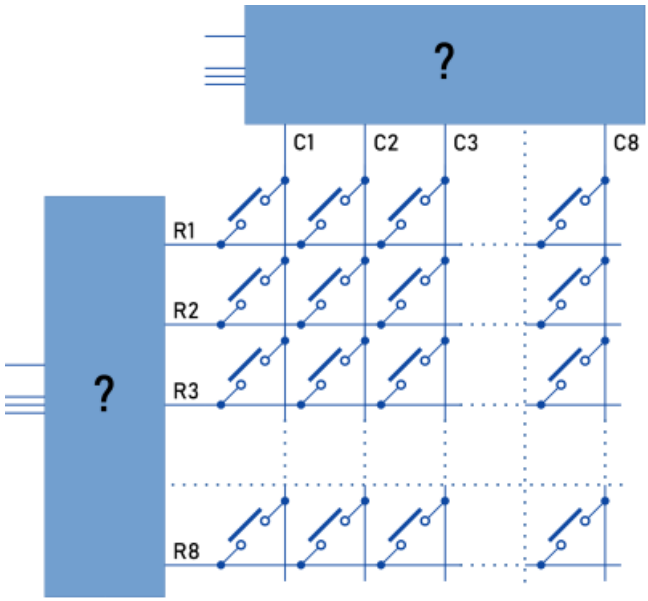
How to manage an input matrix with Multiplexers in standard MobiFlight software

v1.0 - (C) 2022 Giorgio Croci Candiani (@GioCC) for Mobiflight

What is a matrix?

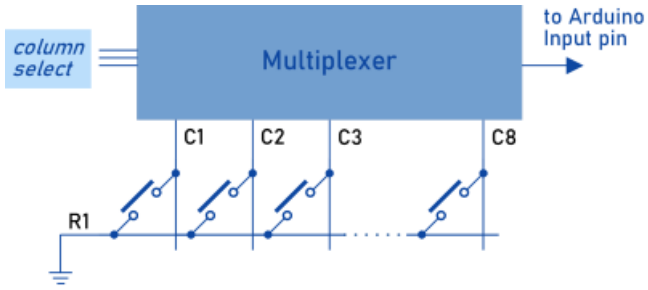
A generic input matrix (for instance an 8x8 keypad) would have a form like in **Fig.1**.



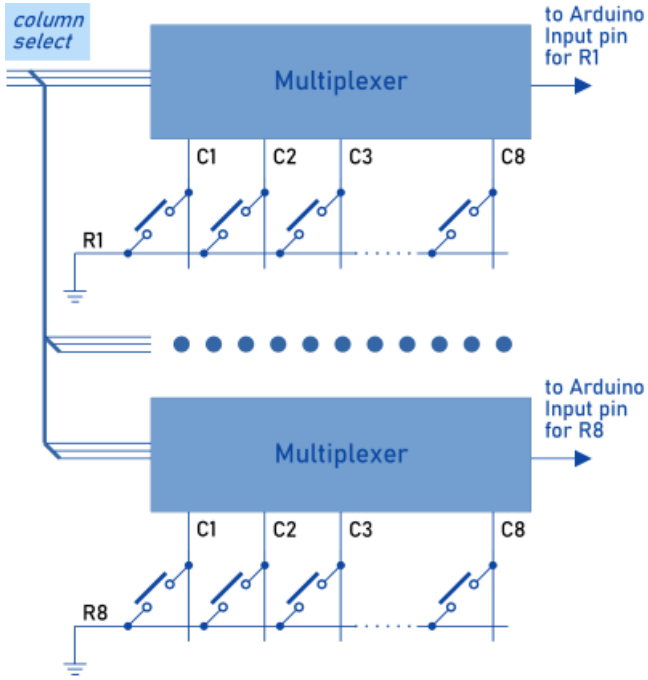
The keys are placed at every crossing between a row line and a column line, and the driver "blocks" could be of very different types - for instance multiplexers, shift registers, dedicated controllers, or a microcontroller itself (like an Arduino board) using its GPIO lines directly.
Regardless of the efficiency of the different solutions (which we won't take into account for now), unfortunately... none of these solutions are natively supported by the MobiFlight firmware (yet!).

How could we handle one?

Let's start by considering a naive, but basic way of reading the keypad switches using a "conventional" connection, the kind normally supported by MobiFlight.
For each row, we could have the keypad buttons, shorting to a common ground terminal, read by a multiplexer that sends the reading to an Arduino pin (**Fig.2**).



This setup is perfectly supported by MobiFlight. Now, to read the full matrix in this way we would just have to repeat the same circuit 8 times (or as many as our row number; **Fig.3**), with one multiplexer and its associated Arduino input for each row.
(To be fair, this is not a "true" matrix - we don't have a grid of column lines - but it's at least very close, so let's pretend it is one).



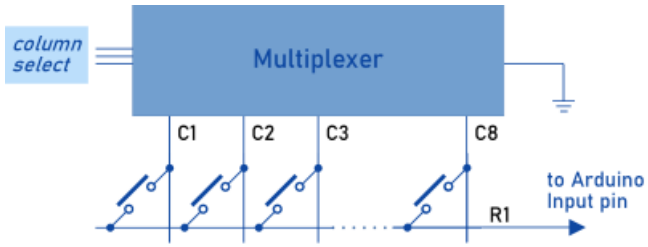
Even though the full set of rows might not be required, that's quite a wiring effort (not to mention the cost and size!).
Neglecting this aspect for a moment, **let's assume that we are happy with this result and start configuring this setup in the MobiFlight Connector**.
To summarize, we will configure one multiplexer (with its assigned Arduino input pin) per row; the inputs of the multiplexers are the buttons of that row, according to their column connection.
Why are we writing a configuration for a setup which is different from the final one? This will shortly become apparent.

Is there any way to improve it?

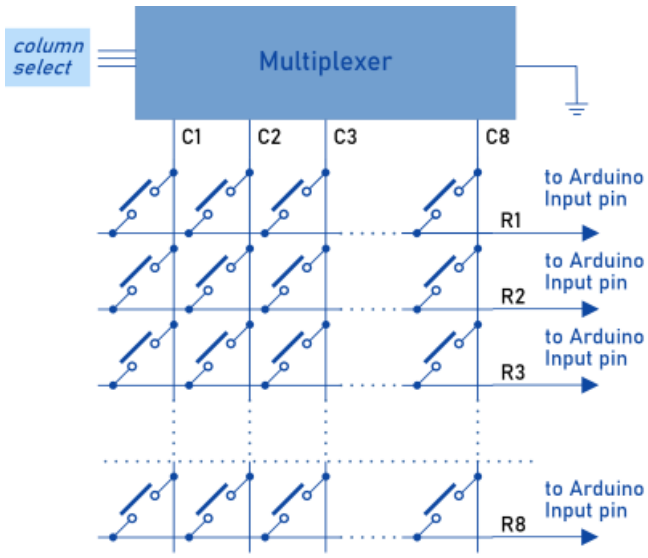
Looking at the base circuit more closely, we could make an interesting observation.
The purpose of the circuit, in the end, is to route one of the terminals (the ground) to the other (the Arduino input).
For a given button press and multiplexer configuration, the two end terminals are simply either connected or not.
If we exchanged the terminals, *nothing in the operation of the circuit would change* - in fact, *we would not even be able to notice there had been an exchange*.

How to exploit this finding?

Let's redraw the circuit with its end connections inverted (**Fig.4**):



we see that now every multiplexer has the common terminal connected to ground: this means that **we could replace the set of multiplexers in *Fig.3*** (which effectively all do the exact same thing!) **with a *single physical multiplexer* connected to all rows of buttons**. Each multiplexer input now "drives" a whole column (**Fig.5**).

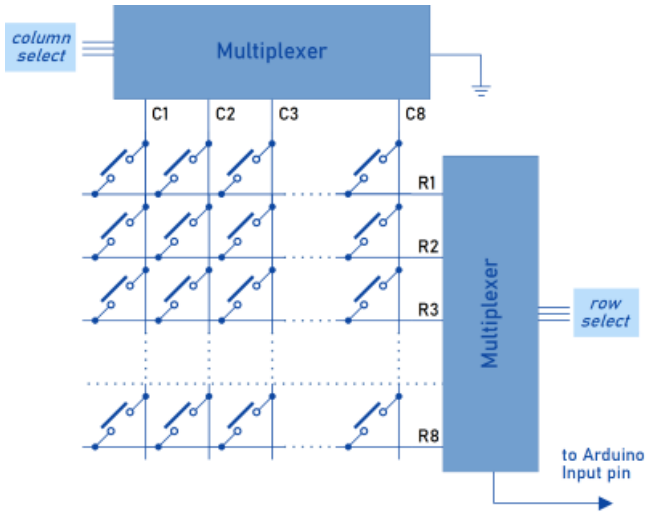


That's already a great result in terms of part count, but there's more: since we saw how these connection changes are *completely transparent* to the Connector configuration, this means that **we can use the very same configuration described before** (which has the distinct advantage of being supported by MobiFlight out-of-the-box)!
In other words, for the Connector we "pretend" to have a set of multiplexers configured as mentioned above, whereas in reality the actual, physical circuit is way simpler (but the Connector doesn't have to know ;).

Is this the ideal solution?

Of course this might not seem the fully optimal solution, because:

- the default multiplexer device in MobiFlight has 8 or 16 lines that are always completely scanned; this means that, if we don't need the full matrix (or most of it), some time is wasted in useless scanning, particularly for small keypads;
- using e.g. a second multiplexer for the row lines (**Fig.6**) might reduce even significantly the number of Arduino pins required (depending on the matrix size);
- a dedicated matrix device driver could improve parameters like scan speed or debouncing.



However, this solution is **immediately available with the current MobiFlight version** (possibly increasing the number of configurable multiplexers, if a larger matrix is required), and some preliminary tests seem to have shown a satisfying performance.

[END]