Four-wire Keypad Interface to Micro-controller

by M. J. Bauer

This article describes a technique to connect a 16-button (4 x 4) keypad to a micro-controller using only four wires. Two of the four I/O pins used on the MCU need to be configurable as either digital or analog inputs. All four pins need to be configurable as digital outputs. The keypad is wired as a 4 x 4 matrix, i.e. 4 rows by 4 columns, with 680 ohm resistors wired between each of the rows and between each of the columns, as shown in Figure 1.

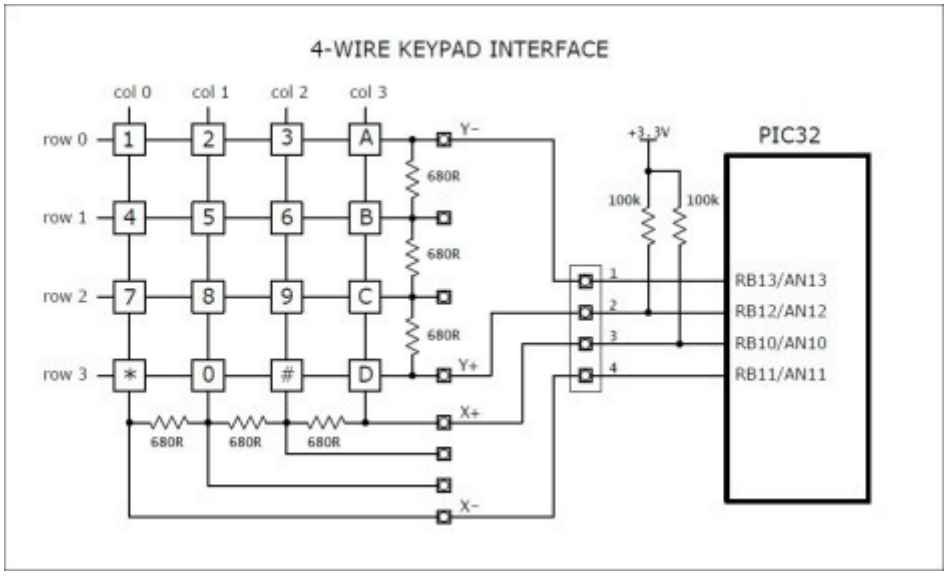

Figure 1

Figure 1 shows how a 4x4 keypad can be wired to a PIC micro-controller using 4 I/O pins: RB10 (X+), RB11/AN11 (X−), RB12 (Y+), RB13/AN13 (Y−). Signals X− and Y− are wired to pins which can be configured as either analog inputs or digital outputs. The resistors' value may be anywhere in the range 200Ω to 1kΩ to suit PIC24 or PIC32MX family MCUs, which specify a maximum source resistance of 3kΩ into ADC inputs. Depending on which key is pressed, there may be up to 3 resistors in series, in effect, when a voltage reading is taken. (See figure 2.) To be on the safe side, a value lower than 1k was chosen, i.e. 680Ω. A value lower than 680Ω may be used, but the circuit would then draw more current from the +3.3V supply than is necessary.

The keypad driver firmware detects when a key is pressed and, if so, which key is pressed and for how long. The technique works for one key press at a time only, i.e. it cannot determine which of two or more keys are pressed at the same time. The technique is very similar to that used to drive a 4-wire resistive touch-screen, except that the X and Y plane voltage resolution is much lower in the case of the keypad.

The driver routine is implemented as a state machine, called periodically, typically at 1 millisecond intervals. The driver is “non-blocking”, i.e. it does not contain wait loops which would delay other tasks. The routine consists of four main “phases”, as follows.

Phase 1: Determine if a key is pressed and, if so, keep track of the time for which it is held down.

Phase 2: Determine which row the pressed key is on.

Phase 3: Determine which column the pressed key is on.

Phase 4: Determine if a “key hit” (event) should be registered and, if so, find the key code at the row and column of the pressed key.

Each of these phases is implemented by one or more states of the state machine. In most phases, more than one state is needed because a delay is required between states, e.g. to allow signal settling after the I/O configuration is changed, to allow the A/D converter time to sample a signal or to perform a conversion. The chosen driver calling interval (1ms) allows ample delay time for these requirements.

In phase 1, pins X+ and X− are configured as digital outputs and set to the low logic level (0V). Pin Y− is configured as an input, i.e. floating, while Y+ is configured as a digital input. While there is no key pressed, a logic high state will be read on Y+ due to the 100k pullup resistor. If any key is pressed, the voltage at Y+ will be pulled down below the logic low threshold. While this condition is maintained, the firmware increments a counter/timer variable every 1 ms. If the key is released, the counter/timer is zeroed.

In phase 2, pins X+ and X− are configured as digital outputs; X+ is set to the high logic level (+3.3V) and X− is set to low logic level (0V). Pin Y+ is configured as a digital input, i.e. floating, while Y− is configured as an analog input. The equivalent circuit of this configuration is shown in figure 2. While a key is being pressed, the resistors between the key columns (X+ and X−) form a voltage divider chain. The voltage at the pressed key, measured on pin Y− (ADC input), is used to find the key column index (0, 1, 2, 3).

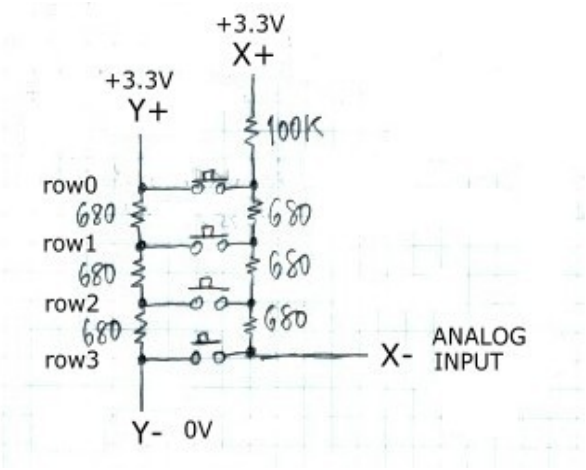

Figure 2

In phase 3, the roles of the columns and rows are reversed. Pins Y+ and Y− are configured as digital outputs; Y+ is set to the high logic level (+3.3V) and Y− is set to low logic level (0V). Pin X+ is configured as a digital input, i.e. floating, while X− is configured as an analog input. While a key is being pressed, the resistors between the key rows (Y+ and Y−) form a voltage divider chain. The voltage at the pressed key, measured on pin X− (ADC input), is used to find the key row index (0, 1, 2, 3).

In phase 4, the firmware checks the “key press counter/timer” value; if a key has been held pressed for more than a certain time (typically 30ms) and a “key hit” has not yet been registered since the last key release, then a “key hit” flag is set. At the same time, the key row and column numbers found in phases 2 ad 3 are used to find the key-code of the pressed key from a look-up table. The key-code is typically an ASCII code representing the symbol marked on the key top, e.g. a digit in the range '0' to '9', letter 'A', 'B', 'C' or 'D', or the '*' or '#' symbol. Of course the key-codes can be customised to suit other keypad symbols.

The “key hit” status flag, key-code and “key press time” can be accessed by the application firmware via function calls: KeyHit(), GetKey() and GetKeyPressTime(), resp. Function KeyHit() clears the “key hit” status flag on exit to ensure that the application does not acquire multiple “phantom” key hits from the same key press event.

A complete source code listing is available for download from the author's website at: www.mjbauer.biz/mjb_resources.htm.