# Low-Power Hex Keypad Using MSP430™ MCUs

## Introduction

Keypads are used in many applications but implementations often struggle to achieve a design that is simple, low cost, and low power. The MSP430FR2000 microcontroller (MCU) is an ultra-low-power device that provides a cost-effective solution using only 512 bytes of nonvolatile ferroelectric random access memory (FRAM). The device's extensive low-power modes enable extended battery life. A keypad design utilizing this MCU can implement a completely interrupt-driven approach that requires no polling and uses minimal external components. While waiting for a keypress, this design consumes only 0.58 µA, and it draws a maximum of only 2.6 µA at 3 V if all keys are pressed simultaneously. The design also takes advantage of the eUSCI peripheral within the MSP430™ MCU to provide a 4800-baud UART interface that reports the button pressed to any connected device. To get started, download project files and a code example demonstrating this functionality. Additionally, the infrared BoosterPack™ plug-in module was used to develop and test this example code with added external pulldown resistors.

## Implementation

This keypad design uses the strategy outlined in Implementing An Ultralow-Power Keypad Interface with MSP430 MCUs. This approach takes advantage of port 1's interrupt capability to wake the device from a low-power mode. The columns of the keypad are connected to P2.0, P2.1, P2.6, and P2.7, and the rows are connected to port pins P1.0 to P1.3. Figure 1 shows these keypad connections to the MSP430 MCU and the associated key numbers.

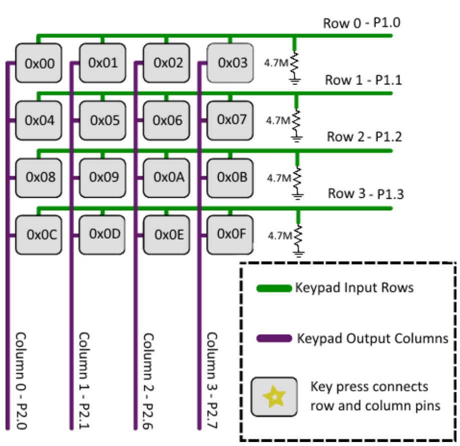



**Figure 1. Keypad MSP430 MCU Connections**

While the MSP430 device is waiting for a key press, it enters a wait-for-press mode where the keypad columns are driven high. Simultaneously the P1.x rows are configured as inputs and pulled low using 4.7-MΩ external pulldown resistors. The device is then put into low-power mode 4, where the current consumption is approximately 0.58 µA, and remains there until a key is pressed.

When a key is pressed, a physical connection is made between a column and one of the P1.x pins. This causes the P1.x pin to interrupt on a rising edge and wake the CPU from low-power mode 4 to continue program execution. First, the key is debounced using the watchdog timer (WDT) for an interval of approximately 15 ms. During this time, the device enters low-power mode 3 to conserve as much energy as possible.

Upon WDT expiration, the device once again wakes from low-power mode and performs a key-scanning algorithm to determine which key is pressed. If a key is pressed, the MSP430 MCU reports the key number using the UART interface.

The device then enters a wait-for-release mode where only one column is driven high. Simultaneously the P1.x rows are reconfigured to interrupt on a falling edge associated with the key being released. This allows the MCU to enter low-power mode 4 while the key is being held and also limits the maximum current consumption to the condition in which all 4 keys on a single column are held down.

When the key is released, it is first debounced using the WDT, and then the key scanning algorithm is executed to ensure no keys are being held. If any keys remain held, the wait-for-release mode continues and the device enters low-power mode 4. Finally, when all keys are released, the MSP430 MCU returns to wait-for-press mode.

The software flow described above can also be viewed graphically in Figure 2. It should be noted that the key scanning algorithm and pin configurations have been optimized for this specific application. If pins are changed, the key scanning algorithm must be changed accordingly.

### Performance

The firmware written for this application was highly optimized for the keypad connections detailed in Figure 1. This was done to ensure it could fit inside a 512 byte memory space while allowing room for slight user modification. It also maintains a high focus on low power, achieving 0.58-µA standby current and a maximum of 2.6 µA when all keys are pressed. This makes the implementation ideal for low-cost battery-powered applications.

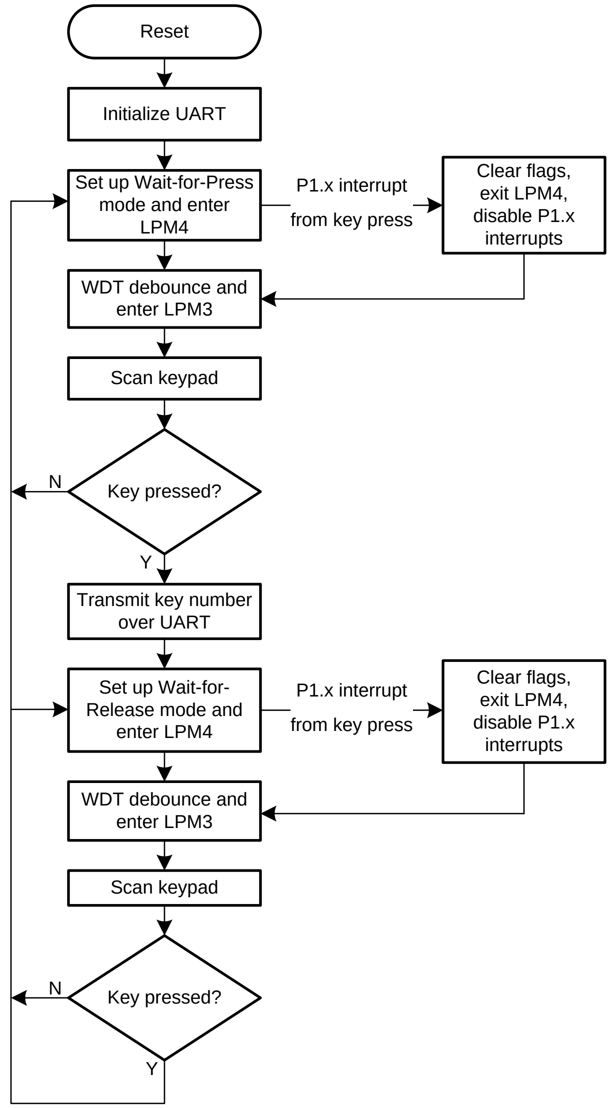
While it may seem like some key presses can be missed using this procedure, the 15-ms debounce interval and interrupt driven approach allow for any button push to be detected. However, this implementation does not take into account ghosting in a keypad matrix, which can only occur when multiple buttons are pressed at the same time. The user can attempt to add a ghost key detection algorithm using the remaining code space available or substitute a larger device if necessary.

### Device Recommendations

The device used in this example is part of the MSP430 Value Line Sensing portfolio of low-cost MCUs, designed for sensing and measurement applications. This example can be used with the devices shown in Table 1 with minimal code changes. For more information on the entire Value Line Sensing MCU portfolio, visit www.ti.com/MSP430ValueLine.

**Table 1. Device Recommendations**

| Part Number | Key Features |
|---|---|
| MSP430FR2000 | 0.5KB FRAM, 0.5KB RAM, eComp |
| MSP430FR2100 | 1KB FRAM, 0.5KB RAM, 10-bit ADC, eComp |
| MSP430FR2110 | 2KB FRAM, 1KB of RAM, 10-bit ADC, eComp |
| MSP430FR2111 | 3.75KB FRAM, 1KB RAM, 10-bit ADC, eComp |




Note: P1.x input must have interrupt capability

**Figure 2. Software Flow**

MSP430 is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

## IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES