Guidelines for use of library Keypad_I2C

G. D. (Joe) Young – July/12, September/13

The Arduino user quickly finds that the number of pins available to connect to external devices the user wishes to control is limited. A 12- or 16-key keypad uses 7 or 8 lines (although these same lines can sometimes be shared—Keypad library, liquid crystal display). Using keypads connected to the inter-ic (IIC or I2C) bus originally developed by Phillips (now NXP) uses only two arduino pins and ground and the I2C bus can have many additional devices also simultaneously connected to the same two lines.

The Keypad_I2C library extends the Keypad library so that a keypad requiring 8 lines or fewer connected to a PCF8574 (8-bit) or PCF8575 (16-bit) I2C parallel-port behaves very nearly the same as one directly-connected to the precious arduino pins. Moreover, two or more such connections can operate simultaneously to permit applications needing more than 16 keys.

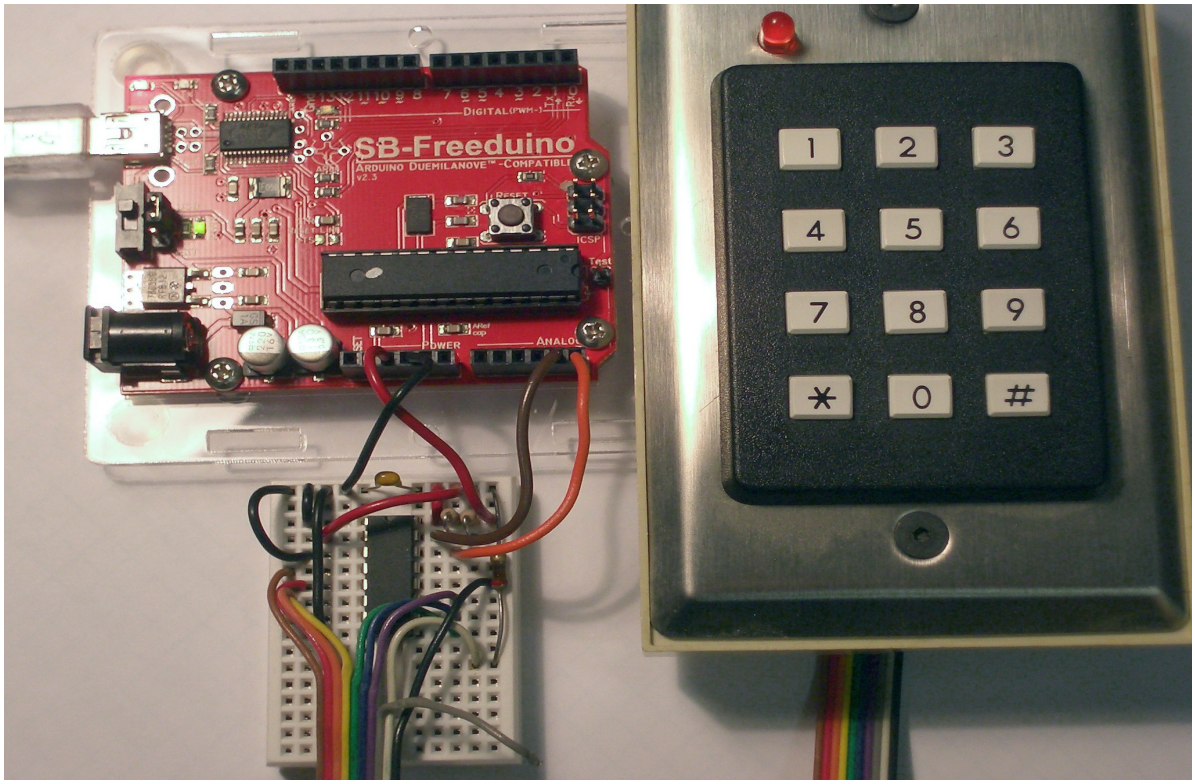
Hardware

For a complete description of the PCF8574, or PCF8575 see the file PCF8574.pdf or PCF8575C.pdf.

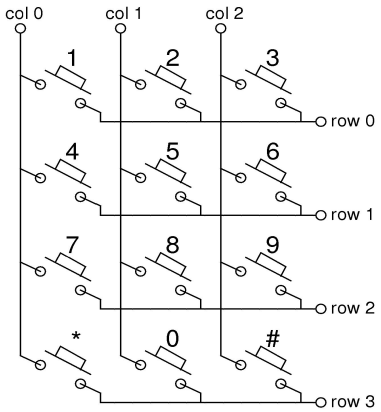
For the wiring diagram and a possible physical layout for an interface to a particular keypad, see the file I2CKeypadDatasheet.pdf.

These files will be found in the Keypad_I2Cdocs folder inside the Keypad_I2C library folder.

Shown here is a photo of a breadboard hookup. Note that it is almost irrelevant how the connections to the keypad are made—they are specified in the software. But it may simplify understanding what is seen initially if the row pins are connected to PCF8574 pins 4, 5, 6, and 7 and column pins to 9, 10, 11, and 12—as assumed in the HelloKeypad_I2C example sketch.



Keypad layout



The software assumes that the key pad has the arrangement of switches shown in the adjacent diagram—that is, each key when pressed joins a unique pair of row and column pins. For example, pushing the 9 key joins column 2 and row 2. The external connections to the keypad are made to the row pins and the column pins. A common labeling of the switches is shown (the "telephone" keypad layout).

Software

The Keypad_I2C library uses the Keypad library and Wire library. The Wire library will be already available in any arduino installation. Place the Keypad folder and the Keypad_I2C folder in the 'libraries' folder within your sketchbook folder. You may need to restart the arduino environment for the additions to show up in the sketchbook/libraries menu.

For a quick start, select the example sketch HelloKeypad_I2C from the Keypad/Examples menu, upload, and then serial monitor. Keys pressed on the keypad should show up on the serial output. A detailed description of the various parts of this sketch follows.

```
#include <Keypad_I2C.h>
#include <Keypad.h>
#include <Wire.h>
```

/*The first three lines make the functions of the Keypad_I2C library available to this sketch. All three of them are necessary. Keypad.h has the principal functionality, Keypad_I2C extends Keypad's input/output operations to make them use I2C, and Keypad_I2C in turn calls on the Wire library to actually perform the communication between the arduino and the PCF8574 (PCF8575).
*/

```
const byte ROWS = 4; //four rows
const byte COLS = 3; //three columns
```

/*Defines the size of the keypad.
*/

```
char keys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
  {'*','0','#'}
};
```

/*Defines the layout of the keypad, and what character is produced by the library when the corresponding key is pushed. For instance, when the key in the second row and third column is pushed, the character '6' is produced. This matrix can be defined with any desired single characters desired.
*/

```
byte rowPins[ROWS] = {0, 1, 2, 3}; //connect to the row pinouts of the keypad
byte colPins[COLS] = {4, 5, 6}; //connect to the column pinouts of the keypad
```

/*Defines how the keypad wires are connected to the bit numbers of the PCF8574. Bit 0 comes out on the IC's pin 4, bit 1 on pin 5, ..., bit 4 on pin 9, ..., bit 7 on pin 12.
*/

```
int i2caddress = 0x22;
```

/* Specify the address on the I2C bus that this particular IC responds to. The PCF8574 or PCF8575 can have hexadecimal addresses 0x20, 0x21, .., 0x27 determined by the connections to the 3 address lines on pins 1, 2, and 3. A second version of the 8-bit IC, PCF8574A is also available and it's addresses can have hex values 0X38, 0X39, .. , 0x3F.

I2C addresses can be somewhat confusing because the least significant bit of the byte going out on the I2C bus that contains the address is the read/write bit . Consequently, addresses are sometimes referred to by the whole 8-bits, and sometimes by just the 7 most significant bits as if they were right-justified—the arduino Wire library uses this second convention.
*/

```
Keypad_I2C Keypad_I2C( makeKeymap(keys), rowPins, colPins, ROWS, COLS, i2caddress, PCF8574 );
```

/*This line calls the Keypad_I2C library's constructor with all of the specifications outlined above to create the object kpd. The first five arguments are the same as for the direct-connection Keypad's constructor, the sixth argument is the I2C address for this keypad, and the last argument specifies the type of port IC. The last argument is optional to be back-compatible with earlier version of the library that supported only the PCF8574 IC.
*/

```
void setup(){
  Serial.begin(9600);
  kpd.begin();
}
```

/*Starts the serial monitor library, specifying the baud rate to be used, and start the Keypad_I2C library. kpd.begin( ) is required to make the library aware of the state of the I2C port the library will use, as well as starting the Wire library..
*/

```
void loop(){
  char key = kpd.getKey();

  if (key){
    Serial.println(key);
  }
} // end of HelloKeypad_I2C sketch
```

/*The loop repeatedly scans the keypad—`kpd.getKey()`—and if a keypress is discovered, the corresponding character is sent to the serial monitor—`Serial.println(key)`.
*/

Comparison with HelloKeypad

The I2C version of the keypad library is very similar to the direct-connection Keypad library. Consequently, if you have already got a sketch going which uses the Keypad library, it can be switched to using the Keypad_I2C library with a minimum of changes:

- Add the two include statements: `#include <Keypad_I2C.h>` and `#include <Wire.h>`
- Modify the pin assignments in arrays `rowPins[ROWS]` and `colPins[COLS]` to conform to the wiring between the keypad row and colum pins and the PCF8574/PCF8575
- Change the constructor statement to `Keypad_I2C` and add the I2C address parameter, and the type of IC2 IC parameter, keeping the object name that is in your already-working sketch. (Here, it's kpd.)
- Add the `kpd.begin()` statement in setup( )

Other Features

The Keypad_I2C library has two functions which will enable some sharing of the I2C port between a keypad that doesn't use all 8 of the PCF8574's i/o pins and other digital i/o. For example, if the keypad is a commonly available 12-key version such as is described for the HelloKeypad_I2C example, then there is one extra pin available on the port chip that could be used—say for driving an LED indicator circuit, or whatever. With the PCF8575, there are 16 i/o pins so there will certainly be some extras.

But keep in mind the drive capability of the PCF8574—it's not as powerful as the arduino's digital output pin drivers. The high output is weakly pulled up, ~0.3 mA, the low output can sink ~10 mA.

Because the I2C port must be written all at once, these functions are needed so that the operation of the keypad can be kept separated from the operation of the spare pin(s).

`pinState_set( )` (no parameters, returns a byte (word if 8575)) is called to get the current state of all 8 bits of the port, and `port_write( value )` writes all 8 bits (or 16 for 8575) of value to the port and returns nothing.

The following example of a function to toggle the spare pin illustrates the use of these two Keypad_I2C member functions:

```
byte portState;                 // to hold local copy of kpd port
const byte ledPin = 0b10000000; // msb to be used for LED drive

void toggleLEDpin( void ) {
  portState = kpd.pinState_set( );  // get current kpd port state into portState
  if( portState & ledPin ) {
    portState &= ~ledPin;
  } else {
    portState |= ledPin;
  }
  kpd.port_write( portState );  // write modified state to port
} // toggleLEDpin( )
```