Lab2

Negin Kheirmand

9831023

This weeks lab is about working with keypad.

IC & Modules I used:

- Arduino Nano
- 4*4 Matris Keypad
- Some 100 Ω resistors & wires

Codes for the Lab:

1. Part1 : sketch_pt1_LED directory
2. Part2 : sketch_test directory
3. Part3 :
    a. sketch_pt3_showNum directory
    b. sketch_pt3_debugedVersion directory

Description:

my key pad is a 4*4 keypad therefore has 4+4=8 pins:

pin1: R1 -> Arduino pin d9

pin2: R2 -> Arduino pin d8

pin3: R3 -> Arduino pin d7

pin4: R4 -> Arduino pin d6

pin5: C1 -> Arduino pin d5

pin6: C2 -> Arduino pin d4

pin7: C3 -> Arduino pin d3

pin8: C4 -> Arduino pin d2

---

to test the keypad run the sketch_test and check the keys, if the

Arduino terminal prints the key pressed then the keypad pinouts and

the connections are as the ones specified above:

warning 1: my keypad looks like this:

```
{'0','1','2','3'},
{'4','5','6','7'},
{'8','9','A','B'},
```

{'C','D','E','F'}

so change the code according if yours it's not like this

warning 2: code will only handle one key pressed at a time.

In the first part of this lab the Arduino is supposed to turn on LEDs according to the number pressed in the keypad, if 1 is pressed 1 LED would turn on, if 2 ... and so on.

but since I have only 4 digital pins left to use (8 already used for keypad) my program will only support 4 LED and therefore when you press 1 to 4 keys LEDs will turn on accordingly and if any other number is pressed the output is written in the Arduino terminal

The pinout for this part of the lab:

the keypad is configured as specified above

the LEDs:

ledPin1 -> D13

ledPin2 -> D12

ledPin3 -> D11

ledPin4 -> D10

---

the second part of the lab is basically the test sketch which outputs the key pressed

---

The last part of the lab is to create a number panel to display the number pressed after using the keypad:

for example, if you press (in order):

first press 1 then 2 and at last 3 the number shown in the terminal would be 123 and them check is more than 9 and then output:

invalid number!

the code for this part is in sketch_pt3_showNum directory

but there is a bug, even though the number is defined as long,

if you press these numbers in order:

12345678987

you would get the output:

-539222901

which is clearly wrong this is because the number 12345678987 does not fit

inside the space of a long, to solve this we could use strings and append the input at the end but since it is specified in the lab description

I will leave this code be and create a new sketch for the debugged version, refer to sketch_pt3_debugedVersion for the string version of the code

Different type of Matris Keypads & How they work:

Each keypad is a combination of **row** and **column** circuits. The buttons themselves are simple switch like closures in these circuits. So they differ in the number of rows and columns, they could also vary in the pinout arrangement, some keypads start the pins with rows and then move on to the columns, in some others the order is reverse and sometimes it's neither since the arrangement of the pin headers doesn't really matter. But the main difference is the keypad technology:

Membrane Keypads:

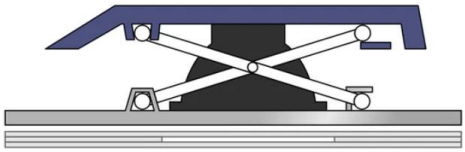
they typically consist of three layers: a top layer with the labels and keys printed on the front; a space layer; and a back layer that features conductive stripes. So when the operator presses down on a key, it forces the upper and bottom layers together. The computer or HMI will then register the keypress.

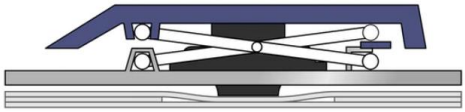
Dome-Switch Keypads:

Another type of keypad that's used widely in commercial applications is dome-switched. Technically speaking, it's a combination of a flat-panel membrane and mechanical-switch keypad. They feature two circuit board traces embedded under a silicone keypad with a dome-shaped switch. These domes create tactile feedback when pressed, making them particularly useful in commercial and work-related applications. Furthermore, dome-switch keypads have a high level of reliability, often lasting for as many as 5 million cycles.

Scissor-Switch Keypad

A third type of keypad is the scissor-switch. It lives up to its namesake by featuring keys that are designed like scissors. The scissor-switch keys contain two separate plastic pieces that lock together like a scissors. It also features a rubber dome like the aforementioned dome-switched keypads, but the scissor design improves functionality by keeping the keys connected together. While there are several different ways to design a scissor-switch keypad, most contain three layers.





Capacitive Keypad

Capacitive keypads use capacitance technology to register keypresses. When you press down on a key, it chances the capacitance of the capacitor pads. They are typically designed with foam element keys that are finished with aluminum. Capacitive keypads aren't as popular as membrane keypads, although they are still a viable choice for certain applications.

Source: http://www.nelson-miller.com/comparing-the-different-types-of-keypad-technology/

SWITCH/KEY BOUNCE:

What is switch bounce? When you push a button, press a micro switch or flip a toggle switch, two metal parts come together. For the user, it might seem that the contact is made instantly. That is not quite correct. Inside the switch there are moving parts. When you push the switch, it initially makes contact with the other metal part, but just in a brief split of a microsecond. Then it makes contact a little longer, and then again a little longer. In the end the switch is fully closed. The switch is bouncing between in-contact, and not in-contact. "When the switch is closed, the two contacts actually separate and reconnect, typically 10 to 100 times over a period of about 1ms. Usually, the hardware works faster than the bouncing, which results in that the hardware thinks you are pressing the switch several times. The hardware is often an integrated circuit.

How to deal with it:

To overcome this problem you may use a low pass filter, you can just out a capacitor to ground to filter the small pulses which follows the main pulse.

Source1: https://www.edaboard.com/threads/what-is-key-debounce-and-how-to-eliminate-it.54631/

Source1: https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/

Small definition of Functions (Library: Keypad.h):

- **Keypad(makeKeymap(userKeymap), row[], col[], rows, cols)**
  - Instantiates a Keypad object that uses pins specified on row[] as row pins, and pins specified in col[] as column pins. The rows and cols variable is the length of each array.
- **Char getKey()**
  - Returns the key that is pressed, if any. This function is non-blocking.
- **Char getKeys()**
  - This function is the multi-keypress version of the getKey() function.
- **char waitForKey()**
  - This function will wait forever until someone presses a key.
  - **Warning:** It blocks all other code until a key is pressed. That means no blinking LED's, no LCD screen updates, no nothing with the exception of interrupt routines.
- **KeyState getState()**
  - Returns the current state of any of the keys.
    The four states are IDLE, PRESSED, RELEASED and HOLD.
- **boolean keyStateChanged()**
  - Let's you know when the key has changed from one state to another. For example, instead of just testing for a valid key you can test for when a key was pressed.

Source1: http://domoticx.com/arduino-library-keypad/

Source2: https://www.arduino.cc/en/Reference/GetKey **(and other functions used)**

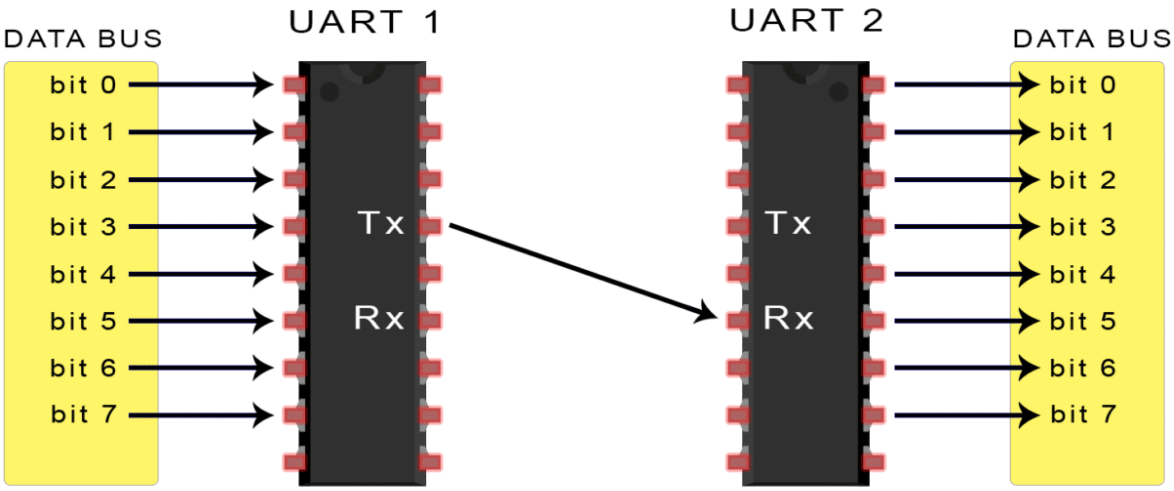
Serial

Description

Used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART), and some have several. On my board(Nano) pins 0 and 1 are used for communication with the computer.

- begin()
  - Sets the data rate in bits per second (baud) for serial data transmission
- end()
  - Disables serial communication, allowing the RX and TX pins to be used for general input and output.
- find()
  - reads data from the serial buffer until the target is found. The function returns true if target is found, false if it times out.
- parseInt()
  - Looks for the next valid integer in the incoming serial. The function terminates if it times out
- println()
  - Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n').
- read()
  - used to Read incoming serial data.
- readStringUntil()
  - reads characters from the serial buffer into a String. The function terminates if it times out
- write()
  - Writes binary data to the serial port. This data is sent as a byte or series of bytes; to send the characters representing the digits of a number use the print() function instead.

UART



In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs.

UARTs transmit data ***asynchronously***, which means there is **no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART**. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits.

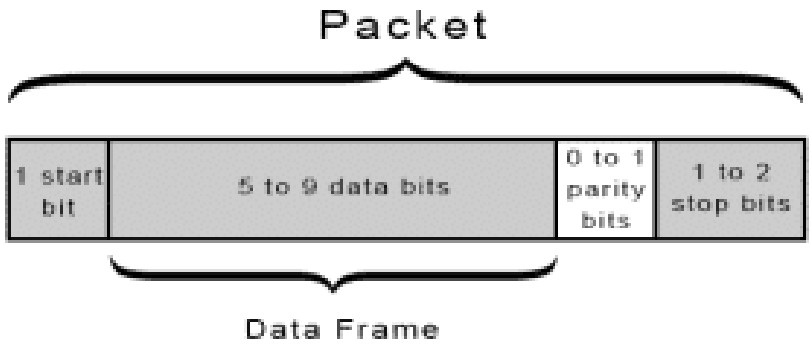
When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the *baud rate.* Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps). Both UARTs must operate at about the same baud rate. The baud rate between the transmitting and receiving UARTs can only differ by about 10% before the timing of bits gets too far off.

Both UARTs must also must be configured to transmit and receive the same data packet structure.

How it works:

The UART that is going to transmit data receives the data from a data bus. The data bus is used to send data to the UART by another device like a CPU, memory, or microcontroller. Data is transferred from the data bus to the transmitting UART in parallel form. After the transmitting UART gets the parallel data from the data bus, it adds a start bit, a parity bit, and a stop bit, creating the data packet. Next, the data packet is output serially, bit by bit at the Tx pin. The receiving UART reads the data packet bit by bit at its Rx pin. The receiving UART then converts the data back into parallel form and removes the start bit, parity bit, and stop bits. Finally, the receiving UART transfers the data packet in parallel to the data bus on the receiving end.

UART transmitted data is organized into *packets*. Each packet contains 1 start bit, 5 to 9 data bits (depending on the UART), an optional *parity* bit, and 1 or 2 stop bits:



Source: **https://www.circuitbasics.com/basics-uart-communication/**

You can also refer to this link to see a more detailed version of each UART communication step.

USART

- **Basically the same as UART just in a synchronous way.**

Like a UART (Universal Asynchronous Receiver/Transmitter), a USART provides the computer with the interface necessary for communication with modems and other serial devices. However, unlike a UART, a USART offers the option of **synchronous** mode. In program-to-program communication, the synchronous mode requires that each end of an exchange respond in turn without initiating a new communication. Asynchronous operation means that a process operates independently of other processes.

Practical differences between synchronous mode (which is possible only with a USART) and asynchronous mode (which is possible with either a UART or a USART) can be outlined as follows:

Synchronous mode requires both data and a clock. Asynchronous mode requires only data.

In synchronous mode, the data is transmitted at a fixed rate. In asynchronous mode, the data does not have to be transmitted at a fixed rate.

Synchronous data is normally transmitted in the form of blocks, while asynchronous data is normally transmitted one byte at a time.

Synchronous mode allows for a higher DTR (data transfer rate) than asynchronous mode does, if all other factors are held constant.

Source: **https://whatis.techtarget.com/definition/USART-Universal-Synchronous-Asynchronous-Receiver-Transmitter**
