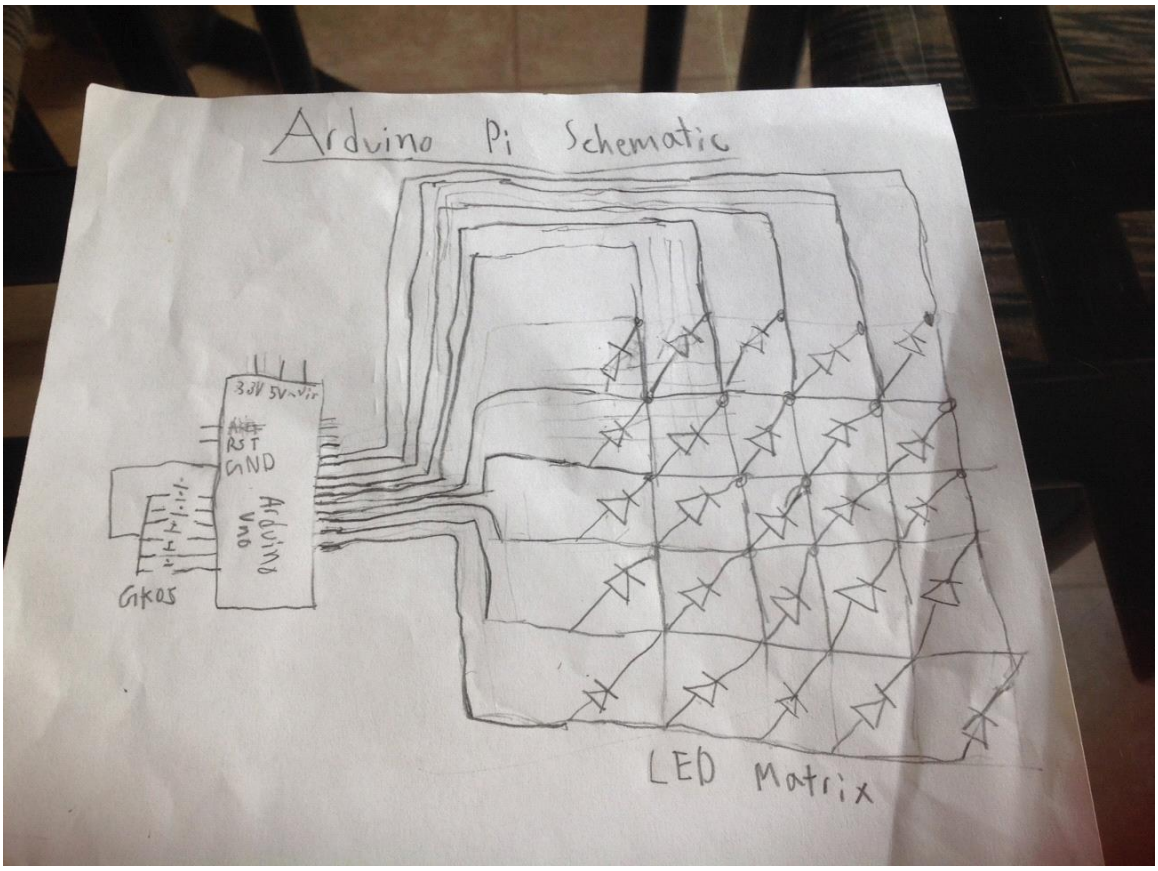
Arduino Pi

The Arduino Pi is a single board computer based on Arduino platform. The Arduino Pi has a six button keyboard known as Global Keyboard Open Standard (GKOS) .

(http://gkos.com/gkos/arduino/index.html), which is a chorded keyboard, and has an LED matrix based display. The LED matrix is a scrolling 5x5 LED matrix. It has 4 digital pins for development using the Arduino platform. However, if the Arduino Pi is developed on a board apart from Arduino Pro Mini, the number of devolopmental digital and analog pins could change. Bitlash was the scripting language for programming externally.

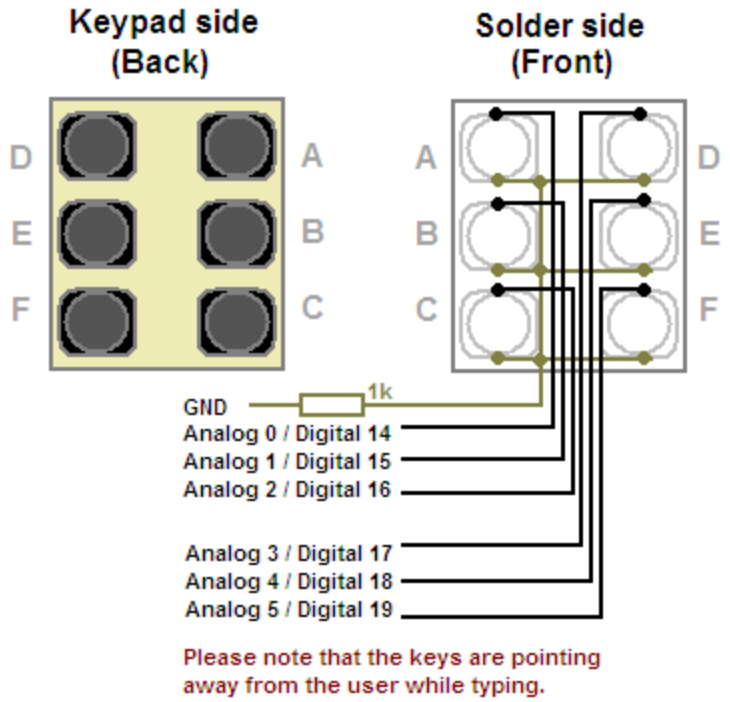
Figure 1:



This is the schematic for Arduino Pi

GKOS Keypad:

With the 6 GKOS keys, directly connected to Arduino I/O pins, it is possible to type any text and enter most functions found on the PC keyboard (Alt, Ctrl, Tab...). The GKOS hardware is very simple, requiring only six pushbuttons to be connected to Arduino pins and to ground. In the Arduino Pi project, the analog pins are used taking up all the analog pins (unless an Arduino board with more than 6 analog pins is used).



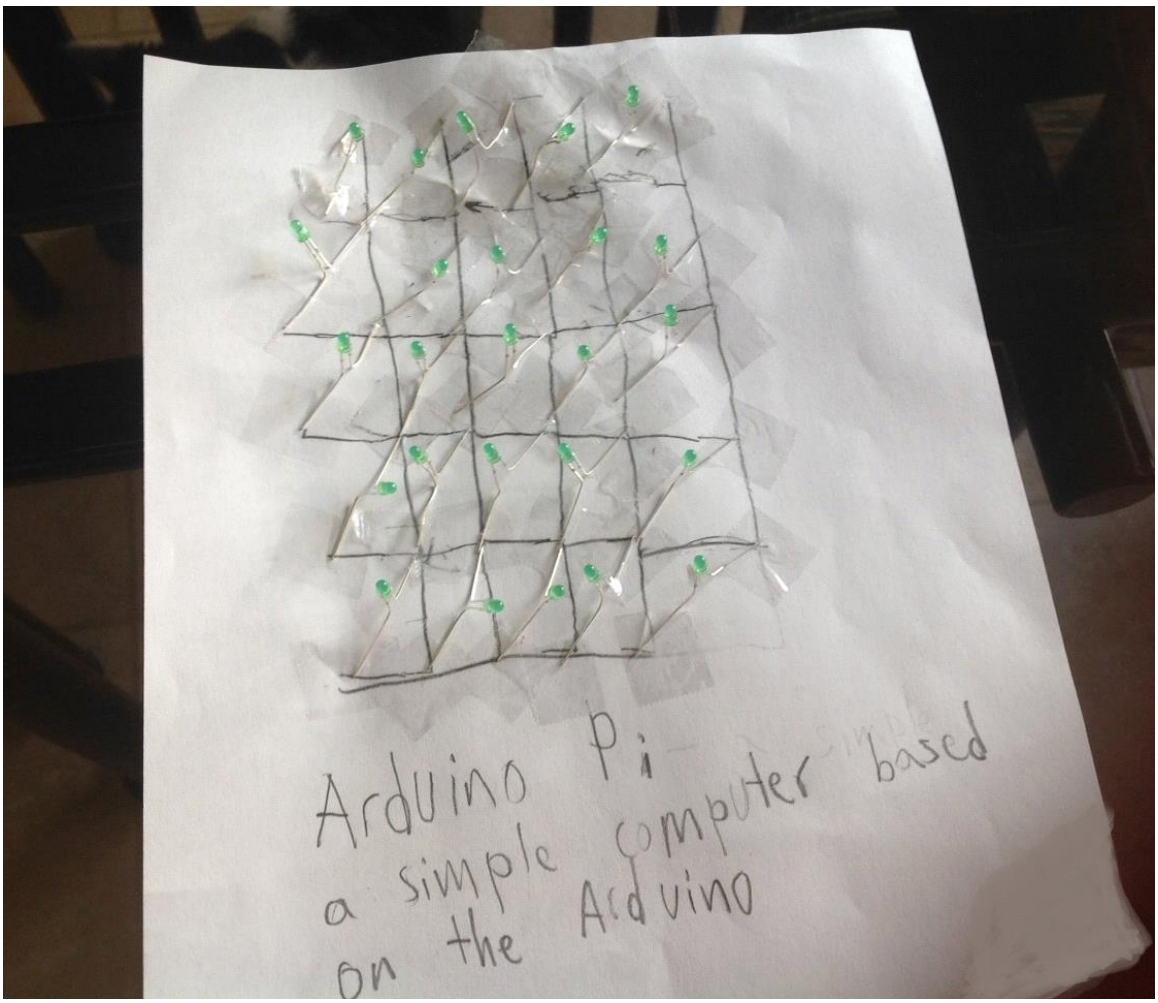
LED Matrix

LED matrices were chosen because they're cheap, compared to OLED and LCDs, simple to make, and easy-to-use. It was important that each of the characters could be understood but easy to interface. 5x5 LED matrices were chosen mainly for size and cheapness. Also, it was possible to reuse code as people have already worked on it and shared their code. The code for the LED matrix mainly came from:

http://www.instructables.com/id/5X5-dot-matrix-on-Arduino-gets-text-from-pc-2/

The LED matrix was hard to build on a breadboard so the LED matrix will be a paper circuit connected to the Arduino on a breadboard. Figure 2 shows the first LED matrix on paper prototype.

Figure 2:



This is the LED Matrix

Bitlash:

When designing a computer, it is important to chose a scripting language, a language that is interpreted by the processor (in this case, the Arduino). I have seen that the most common language is BASIC. BASIC is certainly a great language but it needs lots of code to set it up on the Arduino. However, Bitlash only requires a few lines of code to setup (assuming the Bitlash library is already installed).

Bitlash is an open source interpreted language shell and embedded programming environment for Arduino. The Bitlash shell runs entirely on the Arduino and supports many of the familiar Arduino functions. Bitlash interprets commands you type on the

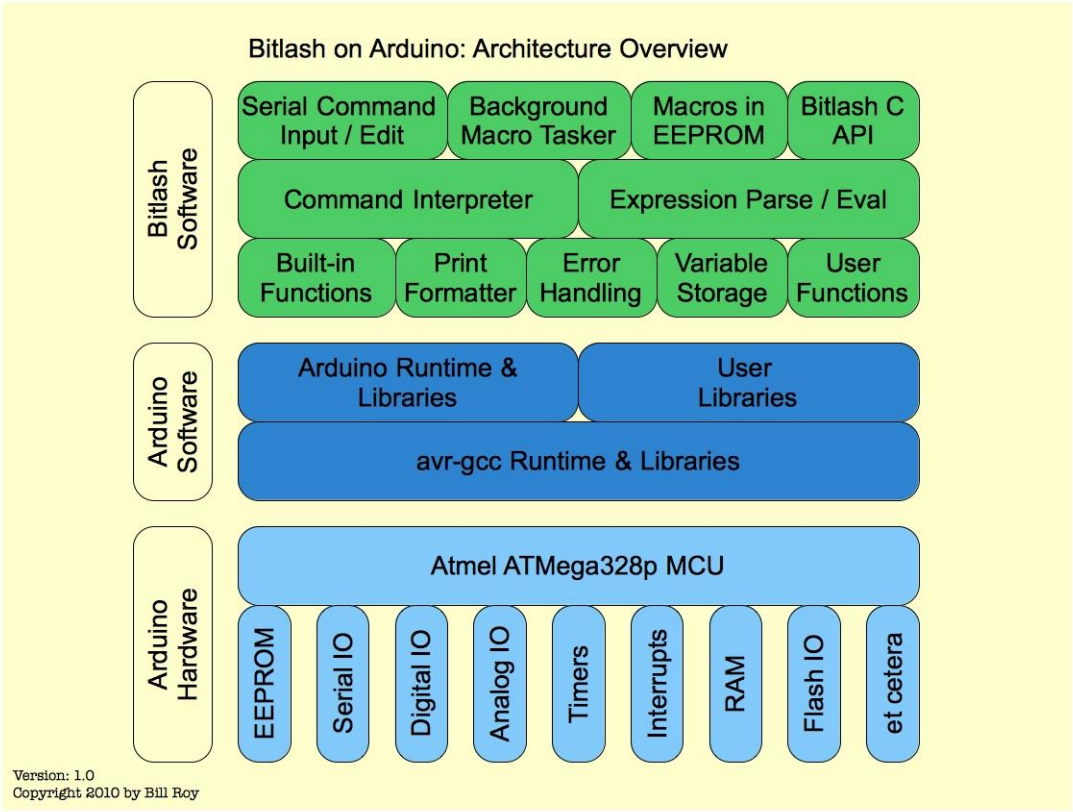
serial port or send from your favorite PC-side programming environment. In this project Bitlash is integrated with the GKOS keypad so instead of using a serial port to enter commands, the GKOS keypad is used instead.

An Arduino C program can interact with Bitlash using some functions listed on https://github.com/billroy/bitlash/wiki/api, two of which are very important for the Arduino Pi.

Bitlash website:

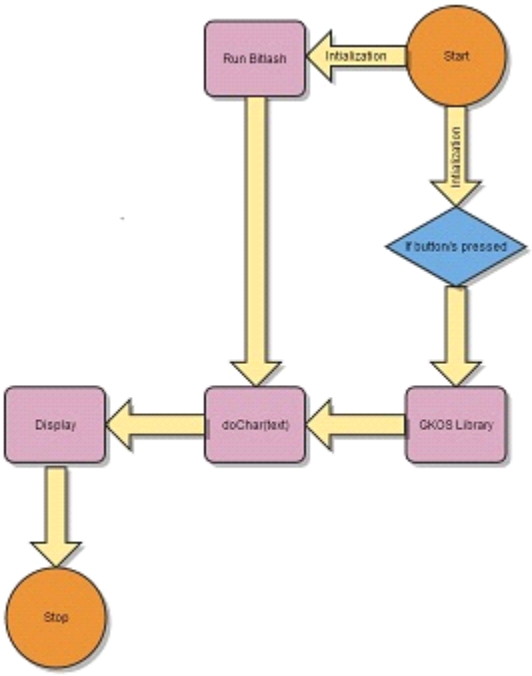
https://www.bitlash.net

Bitlash Architechure:



How it works:

Figure 3: **Figure 4:**



A simple block diagram showing the input and output and how they connect to the Arduino

This flowchart shows how the code operates. doChar(text) represents the Bitlash function that takes in the text returned from processing using the GKOS library.

Figure 3 and 4 demonstrate how the Arduino Pi works.

Modified Code:

```
#include <Gkos.h>

#include "bitlash.h"
```

```
int i;
int rownum[5];
int colnum[5];
B00100};

byte A[] = {
B00100,
B01010,
B11111,
B10001,
B10001};
byte B[] = {
B11110,
B10001,
B11110,
B10001,
B11110};
byte C[] = {
B11111,
B10000,
B10000,
B10000,
//etc..
char* gEntry = "";
Gkos gkos(0, 1, 2, 3, 4, 5);
float timeCount = 0;



void setup() {
```

```
    initBitlash(57600);
    runBitlash();
    setOutputHandler(&serialHandler);
    int rownum[] = {
    0,1,2,3,4 };

    int colnum[] = {
    5,6,7,8,9 };
    Serial.begin(9600);
// initialize the I/O pins as outputs:

// iterate over the pins:
for (int thisPin = 0; thisPin < 5; thisPin++) {
// initialize and turning off colnum pins:
pinMode(colnum[thisPin], OUTPUT);
digitalWrite(colnum[thisPin], LOW);
}

// iterate over the pins:
for (int thisPin = 0; thisPin < 5; thisPin++) {
// initialize and turning off rownum pins:
pinMode(rownum[thisPin], OUTPUT);
digitalWrite(rownum[thisPin], HIGH);
}
}
```

```
void loop(){
    gEntry = gkos.entry(); // Will return empty immediately if no entry
    //String(gEntry);
    char gEntrychar;
    gEntry = &gEntrychar;
    //gEntry.toCharArray(gEntrychar, 1);
    //strcpy(gEntrychar, gEntry);
    runBitlash();
    doCharacter(gEntry);
}



int row(int i) {
if(i == 1) {
return 0;
} else if (i == 2) {
return 1;
} else if (i == 3) {
return 2;
} else if (i == 4) {
return 3;
} else if (i == 5) {
return 4;
}
}

int col(int i) {
if(i == 1) {
return 5;
```

```
} else if (i == 2) {
return 6;
} else if (i == 3) {
return 7;
} else if (i == 4) {
return 8;
} else if (i == 5) {
return 9;
}
}
```

```
void serialHandler(byte b) {
        if (b == 'A') {
            show(A, 500);
        }
                if (b == 'B') {
            show(B, 500);
        }
                if (b == 'C') {
            show(C, 500);
        }
                if (b == 'D') {
            show(D, 500);
//goes on for every character
  }
}
```

```
// routine to show a frame of an image stored in the array pointed to by the image parameter.

// the frame is repeated for the given duration in milliseconds

void show( byte * image, unsigned long duration)

{

unsigned long start = millis(); // begin timing the animation

while (start + duration > millis()) // loop until the duration period has passed

{

for(int row = 0; row < 5; row++)

{

digitalWrite(rownum[row], HIGH); // connect row to +5 volts

for(int column = 0; column < 5; column++)

{

boolean pixel = bitRead(image[row],column);

if(pixel == 1)

{

digitalWrite(colnum[column], LOW); // connect column to Gnd

}

delayMicroseconds(300); // a small delay for each LED

digitalWrite(colnum[column], HIGH); // disconnect column from Gnd

}

digitalWrite(rownum[row], LOW); // disconnect LEDs

}

}

}
```

Arduino Pi receives input through analog pins 0-5 which are connected to 6 pushbuttons. Data is then interpreted by using the GKOS Arduino library. The parsing code is based of one of the example sketches. The command is then used as the variable for the Bitlash function that interprets it. However, GKOS only will return character pressed for every time a user presses the buttons. For example, if a user pressed only

the first button (which is A or 1), then GKOS will only return a. Thankfully, Bitlash has a function for this case. According to the online documentation for Bitlash, "It is also possible to drive Bitlash's internal command line editor one character at a time. This is convenient if you have a character input device."

In the code, the setOutputHandler() Bitlash function is required so that the output of the Bitlash commands will be displayed on the LED matrix. Typically, Bitlash would send the output to the serial console.

The show function is responsible for the controlling of the LED Matrix. It retreives and goes through each of the pixel maps for the characters. It turns on the LEDs either high or low depending on the value of each bit and where the bit is located in the pixel map. It scans rows and turns on the needed LEDs only in one row at time. It goes in a way like the following (from http://www.appelsiini.net/2011/how-does-led-matrix-work):

- Start by having everything disconnected.
- Connect positive voltage all the needed columns.
- Connect row to ground. This lights the needed LEDs in the row.
- Disconnect the row and all columns.
- Do the same steps one by one to all rows and then start from the beginning.

Do this slowly and it would only look like blinking rows.However, when this is done really fast, human eye can see it as one image.

Future of Arduino Pi:

Arduino Pi is still in its early stages and a lot needs to be improved. There are three main problems need to be fixed in the future to improve the Arduino Pi.

First problem that needs to be made is a way to view what a person writes in order for the person to correct the syntax of his commands. However, this will be no use if the user can't even correct his or her own mistakes there is no back button or no cursor to edit the syntax of a command. This is the second major problem that needs to be fixed. A way to parse backspace, scrolling, etc. will need to be made.

Finally, a bigger LED matrix will be needed as not much can be done with a 5x5 LED matrix.