12-Digit Keypad Guide for the BeagleBone Green

by Team Phílos: Răzvan Crețu, Josh Fernandez, Andrew Song, and Mohamed Yahye
Last update: December 4, 2017

This document guides the user through:

1. Understanding the different pins, buttons, and connections on the keypad
2. Wiring the keypad on the breadboard and Zen Cape
3. Running a C program on the target to read values from the keypad

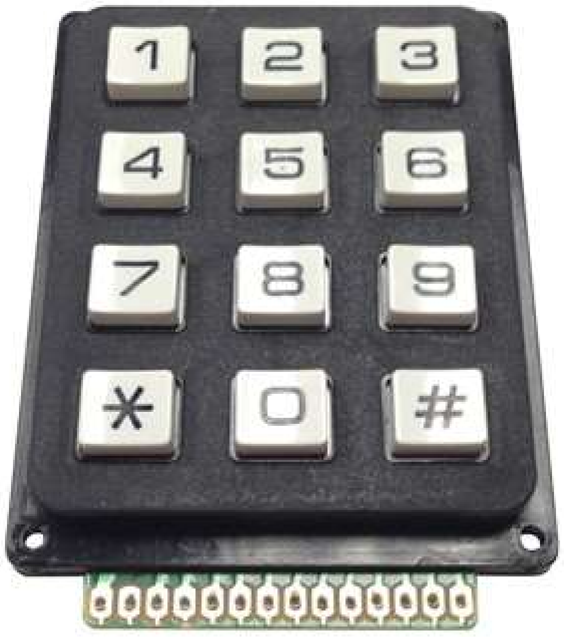
Table of Contents

1. Introduction.......... 2
2. Breadboard Wiring.......... 3
3. Sample Code.......... 6
4. Acknowledgments.......... 9

1. Introduction

This guide will provide step-by-step instructions for using the above 12-key keypad with the Beaglebone Green. Our keypad's stock keeping unit (SKU) is "12KEY" from RP Electronics, found here. We were unable to find any documentation on this item other than what is provided on the product page, so we learned how to use it through trial and error. See Figure 1 for an image of the keypad, showing the buttons and the pins.

Figure 1. The 12-button keypad in question



These are the different pins (from left to right) and their corresponding buttons:

Pin #	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Button	`Input`	-	*	7	4	1	0	8	5	2	#	9	6	3

Notes:
- Pin 1 takes input.
- Pin 2 is unused.
- Pins 3 to 14 connects to their corresponding buttons. The pattern runs from bottom left to top right, moving upwards.

These will be explained more in the next section.

2. BreadBoard Wiring

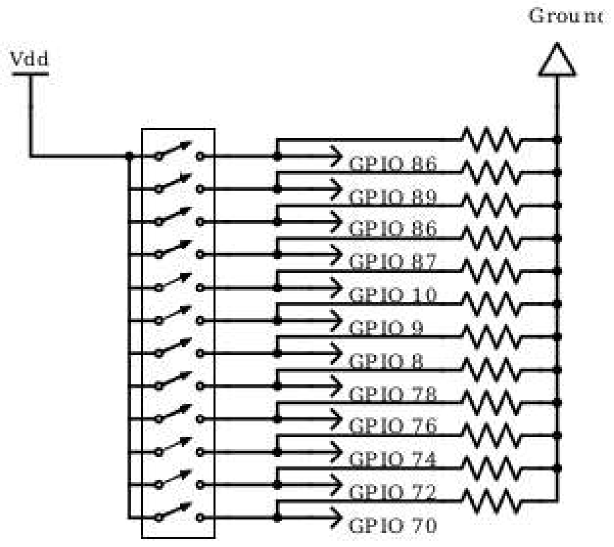
The keypad is rated for up to 24 volts and 20 milliamperes, but the BeagleBone Green can supply only up to 6 milliamperes on a pin.

For our circuit, we used one of the 3.3 volt VDDs as the circuit's power source, one of the ground pins as the circuit's ground, and 12 GPIO pins as inputs.

- ➳ The leftmost pin seen in Figure 1 is the input pin, the second is unused, and the rest are output pins which each correspond to a button.
- ➳ From left to right, the output pins are mapped to the button columns from left to right and down to up (pin 3 is the asterisk button, pin 4 is the 7 button, and so on with pin 14 being the 3 button.
- ➳ The keypad itself uses a common-bus design. It means that, internally, each button acts as a switch between the power bus and its matching output pin. Since this is essentially a series of switches, there are no risks associated with undervolting the keypad (ie. providing 3.3 volts if it calls for 5 volts).
    - ○ However, note that undervolting other devices such as motors may cause them to stall, leading to physical damage. Although we did not have any problems with this, one should be aware in general that if many switches are closed at once, the current will be divided through all the closed switches and may not be picked up by the GPIO pins on the BeagleBone.

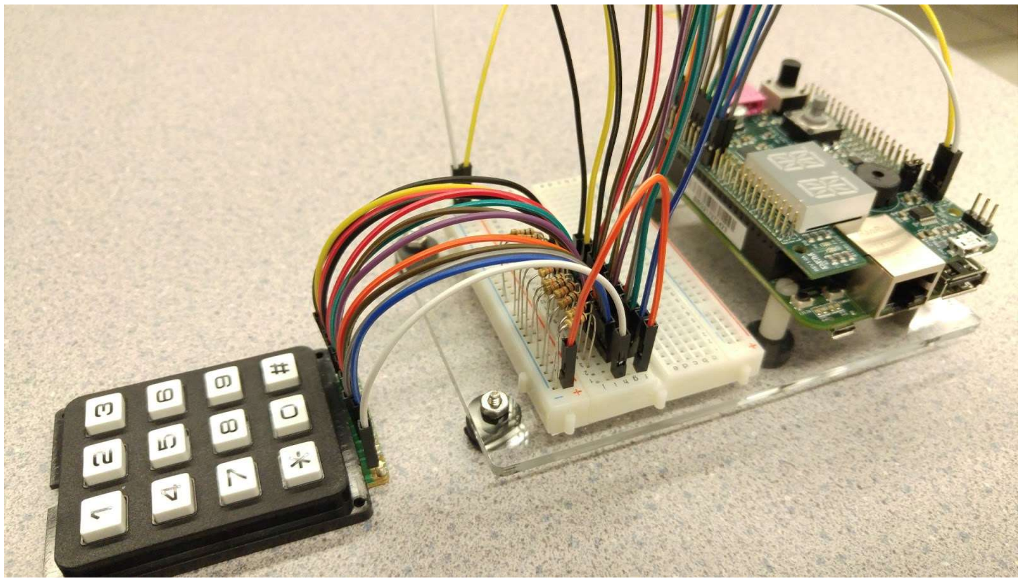
See Figure 2 for a working circuit one could use with the Beaglebone Green. The switch array in the illustration is what the common-bus keypad is internally. The GPIO pins were selected to mostly stay in a single group, but the GPIO pins 36 and 32 are by default in use by another component on the board, so for the sake of simplicity, pins 3 and 4 on the keypad are connected to GPIO pins 88 and 89.

Figure 2. A working circuit diagram



Our implementation of this circuit can be seen in Figure 3. If you wish to use different GPIO pins to read the inputs, be sure to also change the pins used in the sample code later. Remember to check beforehand if they are in use by another component on either the Beaglebone or an expansion cape if present.

Figure 3: Our circuit implementation



In the above photo, the yellow cable is connecting the 3.3 volt VDD to the positive bus on the breadboard and the white cable is connecting the negative bus to the ground.

- The positive bus is connected to the keypad’s input pin through the orange and white cables.
- Each of the keypad’s output pins have a male/male jumper cable soldered on, which are each connected to both the negative bus via 10k ohm resistors and their corresponding GPIO pins via male/female jumper cables.
- The 10k ohm resistors are used as *pull-down resistors*, which keep the readings of the GPIO pins at logical zeroes when the circuit is open (ie. a pin’s corresponding button is not pressed), otherwise the pin is considered *floating* and will read garbage data -- either zeros or ones, the data can’t be trusted.
- Note carefully that Figure 2 and Figure 3 **do not show a current-limiting resistor** (a 470 ohm resistor is sufficient) between the positive bus and the input pin on the keypad, which is needed to limit the current in the circuit; **failure to include this component will damage your BeagleBone** by shorting the VDD to your GPIO pins! The keypad’s internal switches provide marginal resistance to the circuit but should not be relied upon.

3. Sample Code

Below is some basic code which can be cross-compiled and run on your Beaglebone. It will print out the characters of the pressed keypad buttons, if any. If you used different GPIO pins to read the values, be sure to make the appropriate changes to the GPIO array (in this example, GPIO and KEYS must be in the same order). Note that since this program runs indefinitely, it must be killed through the operating system, leaving the GPIO pins exported.

```
// keypad_demo.c

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#define DELAY_NS  100000000L // 100ms in ns
#define BUFFER_SIZE 256
#define NUM_GPIO 12
#define GPIO_BASE_DIR "/sys/class/gpio/"

static const char *GPIO[] = { "88", "89", "86", "87", "10", "9", "8", "78",
"76", "74", "72", "70" };
static const char KEYS[] = { '*', '7', '4', '1', '0', '8', '5', '2', '#', '9',
'6', '3' };

static void init()
{
	FILE *file;
	char buffer[BUFFER_SIZE] = "";

	// export GPIO pins
	for (int i = 0; i < NUM_GPIO; ++i) {
		file = fopen((GPIO_BASE_DIR "export"), "w");

		if (file == NULL)
			perror("Couldn't open GPIO export file!");

		fprintf(file, "%s", GPIO[i]);
		fclose(file);
	}

	// set directions as in
	for (int i = 0; i < NUM_GPIO; ++i) {
		strcpy(buffer, GPIO_BASE_DIR "gpio");
		strcat(buffer, GPIO[i]);
		strcat(buffer, "/direction");

		file = fopen(buffer, "w");
		if (file == NULL)
			printf("Couldn't set GPIO pin %s as input!", GPIO[i]);

		fprintf(file, "%s", "in");
		fclose(file);
	}
}

static void read_values(int *read)
{
	FILE *file;
	char buffer[BUFFER_SIZE] = "";
	char result[BUFFER_SIZE] = "";

	for (int i = 0; i < NUM_GPIO; ++i) {
		// read the value of the GPIO pin
		strcpy(buffer, GPIO_BASE_DIR "gpio");
		strcat(buffer, GPIO[i]);
		strcat(buffer, "/value");

		file = fopen(buffer, "r");
		if (file == NULL)
			printf("Couldn't read GPIO %s!", GPIO[i]);

		fgets(result, BUFFER_SIZE, file);
		fclose(file);

		if (result[0] == '1')
			read[i] = 1;
		else
			read[i] = 0;
	}
}

static void print_results(const int *read)
{
	printf("\nPRESSED BUTTONS:\t");
	for (int i = 0; i < NUM_GPIO; ++i) {
		if (read[i])
			printf("%c ", KEYS[i]);
	}
}

int main()
{
	init();

	int read[NUM_GPIO] = {0,0,0,0,0,0,0,0,0,0,0,0};
	struct timespec delay;
	delay.tv_sec = 0;
	delay.tv_nsec = DELAY_NS;

	while (1) {
		read_values(read);
		print_results(read);
		nanosleep(&delay, NULL);
	}
}
```

Compile this code (one line):

```
arm-linux-gnueabihf-gcc -Wall -g -std=c99 -D _POSIX_C_SOURCE=200809L keypad_demo.c -o
keypad_demo
```

Copy `keypad_demo` to a directory accessible by your BeagleBone, and then run it. For example, if ‘0’ and ‘3’ were pressed, the following would be printed to your terminal:

```
PRESSED BUTTONS:    0 3
```

4. Acknowledgments

This guide is a re-explanation of an already-provided guide on the 12-digit keypad. We felt that the previous guide did not adequately explain the core of using the keypad with the Beaglebone.

We would like to give credit to the "**How to Connect a Keypad to the BeagleBone Black**" by Gary Williams, Marc DeRosier, Conor Brady, and Jong-Ju Park, written for their CMPT 433 class during the Fall 2014 semester.