Final Project - Morse Code GPIO RX/TX

Nickoli Londura (londuran) & Benjamin Martin (martinb3) - Group 7
Fall 2018
CS 444
Oregon State University

December 7, 2018

Abstract

This is the Undergrad Final write-up for group 7 of CS 444. We explore how to build a kernel for the Raspberry PI on the os2 server, as well as learning how to modify the drivers for the GPIO pins on the PI. We can assign 2 GPIO pins to function as a transmitter and a receiver in order to be able to send and receive morse code with any given speed.

1 Write Up

1. What do you think the main point of this assignment is? Being able to build a kernel for the Raspberry PI, and also being able to modify GPIO pins in order to send and receive morse code between other Raspberry PIs with the same functionality. Another point for the project was being able to change the speeds at which the morse code is being transmitted or received individually. It is important to know how to create and modify drivers when working with kernels.

2. How did you personally approach the problem? Design decisions, algorithm, etc. We took some contents from the morse trigger, modified and added a few things, and created our own driver that uses GPIO pins to send and receive morse code. We had to modify the Makefile and Kconfig in order to accommodate our driver when building the kernel. We also had to create a file called `morsegpio.c` which was the morse driver code itself. Our output is almost the same as the trigger, whereas the input checks every 1/4 of elapsed time to ensure the signal state.

3. How did you ensure your solution was correct? Testing details, for instance. We started out by using print statements to see if the kernel was built and running properly. We then realized that we couldn't get anything to print to the console, even though we used the right function, but later on we figured out how to. When it came to testing if the transmitter pin and receiver pin are working correctly, we had to build the kernel every time to check and see if the morse codes for inputs strings were correct, as well as the speeds were reflected by changes requested to the SYSfs system. In order to test the requested pins, we tried different GPIO pins for the transmission and reception to see if the morse code still works. We also tried different strings with various lengths, mixture of upper-case and lower-case letters.

4. What did you learn? We learned how to build a Raspberry PI kernel. Specifically, we were able to create a morse code from any input string and have the transmitter pin send it, while the receiver pin get the code and decode it back into words. We also learned how to have the user change the speed at which the pins send and receive, as well as having the user change which GPIO pins be for transmitting and receiving morse code. Another thing we learned was how to use a character device that allows any kind of input string to be transformed into morse code and back into words.

5. How should the TA evaluate your work? Provide detailed steps to prove correctness.

1. Download the Raspbian OS by going to `https://www.raspberrypi.org/downloads/raspbian/` and choosing "Raspbian Stretch Lite"
2. Download Etcher by going to `https://www.balena.io/etcher/` and choosing one for your OS
3. Flash the Raspbian OS onto the SD card for the Raspberry PI using Etcher
4. Open the root directory of the SD card (by plugging it into its USB thingy and into your machine) and add `kernel=kernel8.img` and `enable_uart=1` to the config.txt file.
5. Ensure that the "code" directory is in the same directory as the "linux" directory. If no linux directory exists, it will get created in the directory above the present working directory.
6. Ensure that the builder script can modify files in the linux directory. Do `chmod +777 -R linux` to ensure that the there are full permissions. Also do `chmod +777 -R code` to ensure that everything can run and be moved around in the code directory that we have included.
7. Run the "builder" script to build the kernel. If you get any prompts from the patcher, answer no ("n"). This means that you have previously built and are trying to build again, but the patcher will think to reverse the changes.
8. Ensure that the `.config` file has the appropriate resource written to it.
9. Do `make -j4 ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- menuconfig`
10. Go to `device drivers -> Character devices -> CLICK ON ME BOI` and hit "Y" to select the option. It should have a * next to it. Then, hit "Save", and hit "ok". Then, press exit until you are out of the menu screen.
11. Then run `make -j4 ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- all` again.
12. Copy the "kernel8.img" to the SD card

NOTE: THERE ARE NO PATCH FILES. We wanted you guys to be able to see all the source code, so we have the four files that were pertinent to the project, and nothing else. The builder script will handle (almost) everything.

To test our project quickly, you can do the following:

1. Hook up a female-female jumper wire to GPIO pins 23 and 24. By default, the OUT pin is 24 and the IN pin is 23.
2. Boot up the Kernel. Log in.
3. Do `sudo su` so that you are the superuser.
4. Do `echo [your-word-here] > /dev/morsegpio`
5. Do `cat /dev/morsegpio`

You should see your word printed out over and over. To stop reading, hit CTRL + C. To change the word that is being transmitted, echo a new word to /dev/morsegpio. Other notes of the features:

- We use SysFs for the options.
- To change the input speed reception, do `echo [speed-in-ms] > /sys/class/char/morsegpio/in-speed`. Please keep the input speed a multiple of 4. (This is for Nyquist reasons). By default, the speed is 100ms.
- To change the out speed reception, do `echo [speed-in-ms] > /sys/class/char/morsegpio/out-speed`. By default, the speed is 100ms.
- To change the input pin, do `echo [pin #] > /sys/class/char/morsegpio/in-pin`
- To change the output pin, do `echo [pin #] > /sys/class/char/morsegpio/out-pin`
- A "?" symbol may be printed early on, which indicates an unrecognized character. This may happen due to the read starting in the middle of a morse code being transmitted. The morse code will be transmitted ceaselessly.

Please contact us at martinb3@oregonstate.edu and londuran@oregonstate.edu for any questions regarding setting up our project or issues.

2 Version Control Log

acronym	**meaning**
V	`version`
tag	`git tag`
MF	Number of `modified files.`
AL	Number of `added lines.`
DL	Number of `deleted lines.`

V	tag	date	commit message	MF	AL	I
1		57351b4b4272e2e202f9c37236e6aa529d57d48d 2018-10-04	Initial test	4	118	
2		457cb43d9a5ffd145692bf8048d9f67837ef5b67 2018-10-04	Create README.md	1	1	
3		a0d31694c3643232027d1603ce90157f0aa4f7d0 2018-10-04	Create README.md	1	1	
4		7ed32fd0a03d4cd228a052dc6e82d18036585619 2018-10-04	Small change	0	0	
5		789c79dd6550916482399238babfdb32c9a9bc26 2018-10-04	Added assignment folders	6	111	
6		73e30d4da7ba8a16076476ad9dbc6762dabe5842 2018-10-04	Added assignment folders	6	111	
7		f61afcd4df3637188959edd4269d9964eb01196a 2018-10-04	Organized	10	4402	
8			0	0	0	
9			0	0	0	
10			0	0	0	
11		fa83ec19fc9d6e1c06b20b5edc4ee75f4a0d7b3a 2018-10-04	There we go	0	0	
12		06539913bd0fab683fe6cde3bd06cde97e9431f3 2018-10-10	Added concurrency code	1	35	
13		b5310109d285d732ccc2527efe50f124dc0415bc 2018-10-15	Added stuff	6	57	4

Please continue at the next p

V	tag	date	commit message	MF	AL
14		840df6290e155004b32584878ee1d12810aeff5e 2018-10-15	Finished concurrency assignment	11	2102
15		35987c988887860b9d57168407068aa8864ea9d6 2018-10-15	Changed int in main and spaced code	1	19
16		50b05095e10db4eee4bbd92773c721d7803f9f2d 2018-10-25	Added writing and things	13	505
17		e0c136046374c324bbd47f0723b1055ea4290abe 2018-10-25	modified code	6	477
18		a87023fe93a19dbfe84e84a7374ded43bb4662db 2018-10-25	Finished concurrency project	2	156
19		9ff7580e7ac32d8522936600a6cd6650cda6d7b5 2018-10-29	made the writeup latex	6	6829
20		9036481e1bcef26af90e9b91e33e6f562c8177c1 2018-10-29	Mostly done?	10	670
21		4729168de8c9cce8acb77f5b9baf6c5713864d04 2018-10-29	After assignment 2	14	2830
22		5f4d44aba72909a2a4a9d072166de25b30111c24 2018-11-12	Moved code and patch files into repository	4	506
23		7ec298a2f7318ab4eb0c183c2af6aa13497319d7 2018-11-12	made latex file	4	511
24		7295b1e68a2f97aca4fdde5a2d4fbab20bbbd444 2018-11-12	Added how to tar	14	2850
25		1971a4e862078681cf793638f2f03a4d4399ea94 2018-11-26	Assignment4	7	675
26		4dff62aebc397569fb1cc08e83158cb68968cd0e 2018-11-26	Hi there	13	3466
27		303c70534243944db9832f133b85a5d4dba94b02 2018-11-26	concurrency3	18	1518
28		4a2da519bd67fc5da8f16aa76419e0b634967e87 2018-11-26	deleted unecessary tar	1	0
29		b81d73e61646ec9c901768385dcd988c469fc6c7 2018-12-04	Fixed it	3	42
30			0	0	0

3 Work log

The majority of the project was completed on Tuesday, December 4, 2018. We saved the writeup and the testing on Thursday, December 6, 2018. That made our work log real spicy for the first day we worked on it. Nickoli did most of the work for the write-up and Ben did most of the coding for GPIO driver. This worked out really well because Ben hates writing. Our group will continue this work pattern in the future. Our original group was a group of two, so we decided to work as group 7 again for this project.