fnoble / CC-Bootloader
forked from swift-nav/CC-Bootloader

Watch 2 Star 17 Fork 20

Code Pull requests 0 Projects 0 Security Insights

USB bootloader for the CC1111 RF SoC

51 commits 1 branch 1 release 1 contributor GPL-2.0

Branch: master New pull request Find file Clone or download

This branch is even with swift-nav:master. Pull request Compare

fnoble Remove HEX files for development version. Latest commit 64d092e on 23 Jun 2011

driver	Initial import of device driver inf file for Windows	9 years ago
example_payload	Remove HEX files for development version.	9 years ago
src	Default to timer off.	9 years ago
.gitignore	Added gitignore files for the sdcc intermediate and output files.	9 years ago
LICENSE	Initial import of GPLv2 license document	9 years ago
Makefile	Added a simple hardware abstraction layer to make porting	9 years ago
README.markdown	Added a simple hardware abstraction layer to make porting	9 years ago
bootload.py	Return code not given here so don't check for it.	9 years ago
start.asm	Interrupt forwarding code should now be working, needs further testing.	9 years ago

README.markdown

CC Bootloader

CC Bootloader is a simple USB bootloader for the CC1111 microcontroller (and probably others in the family e.g. CC2511). It allows the update of the microcontroller firmware over its USB port. It also supports reading and writing to the flash memory of the microcontroller e.g. for user data storage.

The bootloader consists of two components, a piece of firmware that is flashed onto the device and a python utility for downloading code and manipulating the flash memory.

Usage

The first step is to flash your device with the `CCBootloader.hex` firmware file. Once the device is flashed with this firmware it will identify itself over USB as a CDC-class device which means it should show up as a serial port without requiring any drivers on Linux and Mac OS X. Windows users should use the `CCBootloader.inf` driver file in the `driver` subdirectory.

If you have downloaded a release version of CC Bootloader it will include a `CCBootloader.hex` file that you can simply flash onto your device. If you have downloaded a development version then see the Building section below to build the `CCBootloader.hex` file. You may also want to build your own version to enable or disable certain features.

You can now use the bootloader.py python script to download your payload code. This script requires the pySerial library. If you have `easy_install` then installing pySerial should be simple:

```
sudo easy_install pyserial
```

For usage instructions of this script, run the script with no arguments:

```
./bootloader.py
```

Preparing your user code for usage with the bootloader is very simple. All you need to do is set your linker to start the code section at a higher location in memory specified by the `USER_CODE_BASE` define in `src/main.h`. For an example of this see the `Makefile` file in the `example_payload` subdirectory. This is the relevant line:

```
LDFLAGS_FLASH = ... --code-loc 0x1400 ...
```

Building

This requires sdcc (Small Device C Compiler). Then it should be as simple as issuing

```
make
```

from the root directory of the project.

Build Options

The only option that can currently be enabled or disabled is weather to start running the user payload code after a certain timeout with no command issued over USB. Once a USB command is issued this timeout will be canceled and a command must be issued to explicitly start the user code. This option is specified in `src/main.h`. Just comment or uncomment this line:

```
#define TIMER
```

You may also change the length of the timeout (default 10 seconds):

```
#define TIMER_TIMEOUT 229 // 10s timeout
```

Please note that if you make changes to the bootloader you may need to adjust the value of `USER_CODE_BASE`. You will need to do this if the linker complains that it has run out of space. This value muse be a multiple of 1,024 so that the user code begins on a page boundary.

You must reflect this change in several places:

1. Change the value of `USER_CODE_BASE` in `src/main.h`
2. Change the value of `--code-size` in `Makefile`
3. Change all the lines similar to `ljmp #(0x1400+0x03)` in `src/start.asm`. The constant `0x1400` should be changed to match `USER_CODE_BASE` but the `+0x??` part should be unchanged.

Hopefully step three will not be needed in the future when I find a better way to implement this part of the code.

If you want the bootloader to only be invoked under certain conditions, e.g. the presence of USB power then please modify the `want_bootloader` function in `main.c`

All the board specific code is in `hal.c` so make sure to take a look at that if you are porting the code to a new board.