# USING OPENLST:

## Getting Started:

Firstly I read the user guide given in Openlst repo, starting with what you will need and everything. Since we are using cc1110, and we have cc1125 evaluation board in lab. FIrst was figuring out similarities and differences between them.
We went with cc1110 because it was used in openlst hardware so more data was available on it.

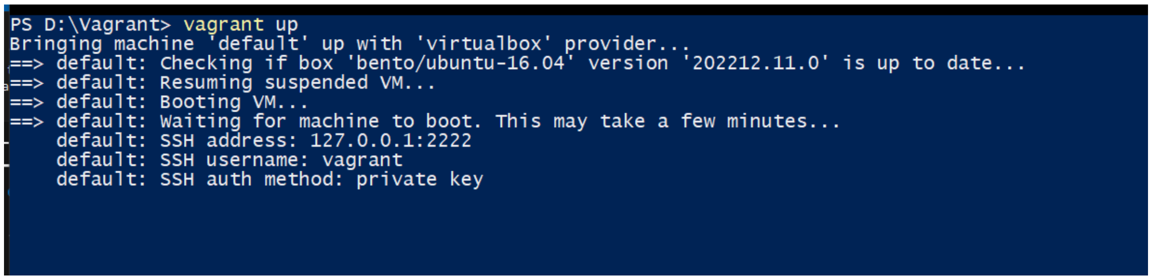
### Use of ccdegubbertool:

We are not sure if evaluation board actually contains a cc debugger tool or not but we definitely need a debugger for our board, as was needed in openlst. For communication with cc, this tool is mandatory.

### Vagrantfile:

I tried to install a vagrant file in my laptop but i faced some errors to get into the virtual environment of the vagrant file given in the openlst documentation, i believe either it is the fault of my laptop or the vagrant file is corrupted. I did't have to research much on it because easy solution is using linux and downloading everything present in the vagrant file directly.
There is a Vagrantfile included in the sample project that creates a VM that includes:

1. cctool (TI CC Debugger support software for programming)
2. SDCC (the compiler toolchain)
3. The included python tools for programming, signing, and testing the application
4. USB passthrough support for the TI CC Debugger and evaluation board serial ports
5. A shared folder with the host machine so you can use development tools on your host while running the toolchain and programmer tools in the VM



```
PS D:\Vagrant> vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'bento/ubuntu-16.04' version '202212.11.0' is up to date...
==> default: Resuming suspended VM...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
```

This is what vagrant means, basically virtual machine is like a wsl in which we can get into some other environment. But i wasn't able to access the openlst's file.

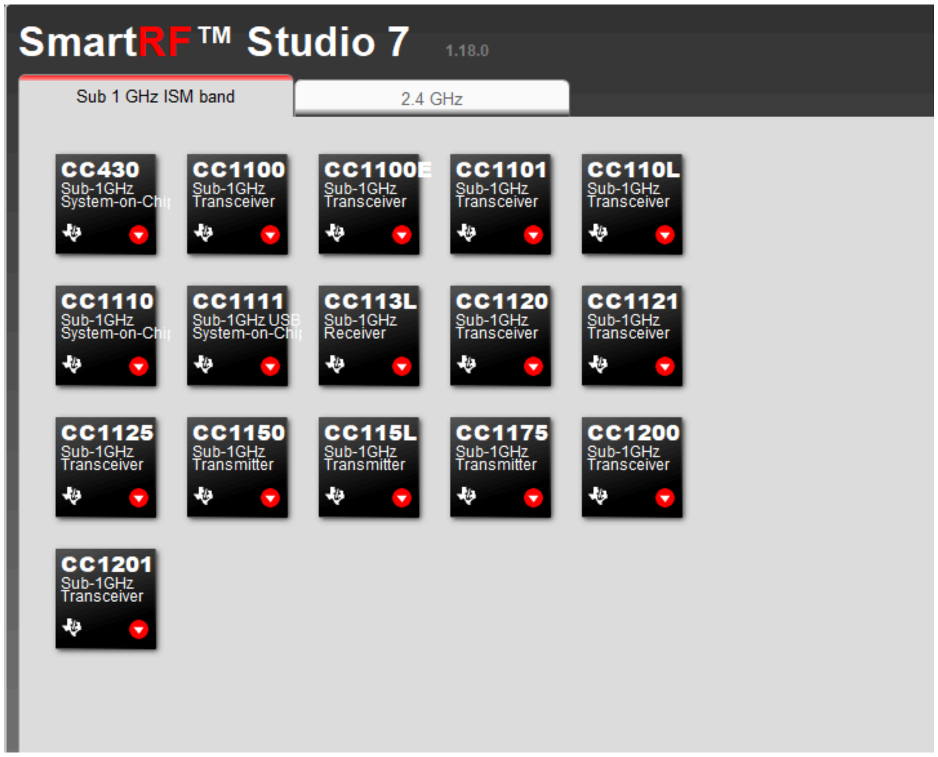
Alternatively, as given in file i setup cctool and sdcc on linux directly.

P.S: I think the evaluation board doesn't contain any cc debugger as shown by the cc tool installed.

## SMART RF STUDIO:

As given in the data sheet of cc1110 and cc1125 chips, smartRF studio can be used for establishing communication between 2 cc, using two laptops and enabling transmission and receiving in each of them.

I was successfully able to use smart RF studio to transmit telemetry data liike text and bits from one cc1125 evaluation board to another in the lab.


SmartRF™ Studio 7
1.18.0
Sub 1 GHz ISM band
2.4 GHz
CC430 Sub-1GHz System-on-Chip
CC1100 Sub-1GHz Transceiver
CC1100E Sub-1GHz Transceiver
CC1101 Sub-1GHz Transceiver
CC110L Sub-1GHz Transceiver
CC1110 Sub-1GHz System-on-Chip
CC1111 Sub-1GHz USB System-on-Chip
CC113L Sub-1GHz Receiver
CC1120 Sub-1GHz Transceiver
CC1121 Sub-1GHz Transceiver
CC1125 Sub-1GHz Transceiver
CC1150 Sub-1GHz Transmitter
CC115L Sub-1GHz Transmitter
CC1175 Sub-1GHz Transmitter
CC1200 Sub-1GHz Transceiver
CC1201 Sub-1GHz Transceiver

### How to use:

On how to use smartRF studio is very easy and is given in one of  the documentation which i attach at the end of this file.
Refer to this link: Texas Instruments CC1310 LaunchPad - SmartRF Radio Configuration

## BOOTLOADER:

### What is a bootloader?

A bootloader is a small program that initializes hardware and loads the main operating system or firmware into the memory of a device.

### Parameters while loading a bootloader:

You will need to specify two parameters when loading the bootloader:

A hardware identifier (HWID): This is a two-byte identifier, typically expressed as a four-digit hex number like "01AB". Some important rules and guidelines apply:

- Hardware IDs 8000 and above are reserved for ground/base station radios
- Hardware IDs 7FFF and below are reserved for remote radios.

- When deployed in the field, it is important that your HWIDs do not overlap with other users of the same frequency band and modulation settings. If you share allocation and are worried about overlap in HWIDs, it is recommended that you coordinate with your issuing authority and contact Planet.
- HWID 0000 is reserved for broadcast messages and should not be used
- HWID FFFF is reserved for local messages sent over the serial port to the ground/base station radio and should not be used
- For testing purposes HWIDs 0001-00FF are recommended. For this tutorial you can just use 0001.

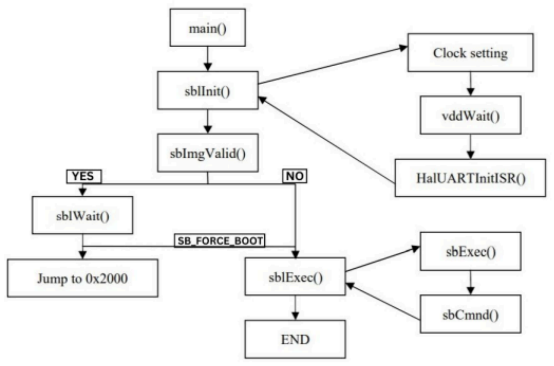
Signature keys for application payloads: The bootloader requires three AES-128 (16 byte) keys. These are used to verify the authenticity of application images. Applications must be cryptographically signed with one of these three keys to run. Keys can either be specified at the command line (as 32 character hex strings) or from a key file (as three comma-separated 32 character hex strings).

- In production, all three keys should be set to non-trivial values (not all zeros or all ones).
- It is okay to set two or three of the keys to the same value if you don't need three distinct keys. In production it is recommended that you have at least two unique keys in order to have a backup in case the primary key is lost.
- For lab testing you can set all three keys to all ones (a value of FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF)

## Bootloader firmware:

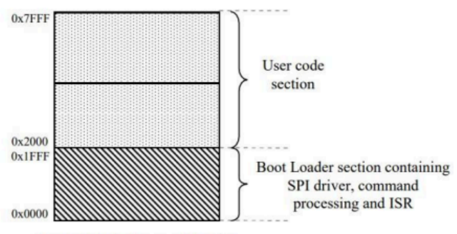
The bootloader code is very complex and uses multiple header files with different segments:
The flowchart on how the bootloader code works is as following:

**Bootloader:** The bootloader for the CC1110 is a program that enables firmware updates by managing the process of receiving new software. The flow-chart for the bootloader code is as following:



Every command explained here:

- **sblInit()**: Sets up the clock, USART, I/O ports, and DMA of the SoC.
- **sbImgValid()**: Checks if the user code in flash is valid.
- **sblWait()**: Waits ~1 minute for a command from an external host (SB_FORCE_RUN & SB_FORCE_BOOT)
- **sblExec()**: Runs the bootloader routine
- **HalUARTUnInitISR()**: Resets all USART-related interrupts and configuration registers.
- **vddWait()**: Checks for a specific voltage level
- **sbCmnd():** Processing commands received from the external host
- **SB_FORCE_RUN** command can be generated by 0x07
- **SB_FORCE_BOOT** command is given by passing 0xF8.


0x7FFF
User code section
0x2000
0x1FFF
Boot Loader section containing SPI driver, command processing and ISR
0x0000

Internal Flash Memory Mapping

---

This is the basic code structure of bootloader code as given in the openlst repo.