



The Journey of Google Taiwan

Yi-Yang Lin

About me



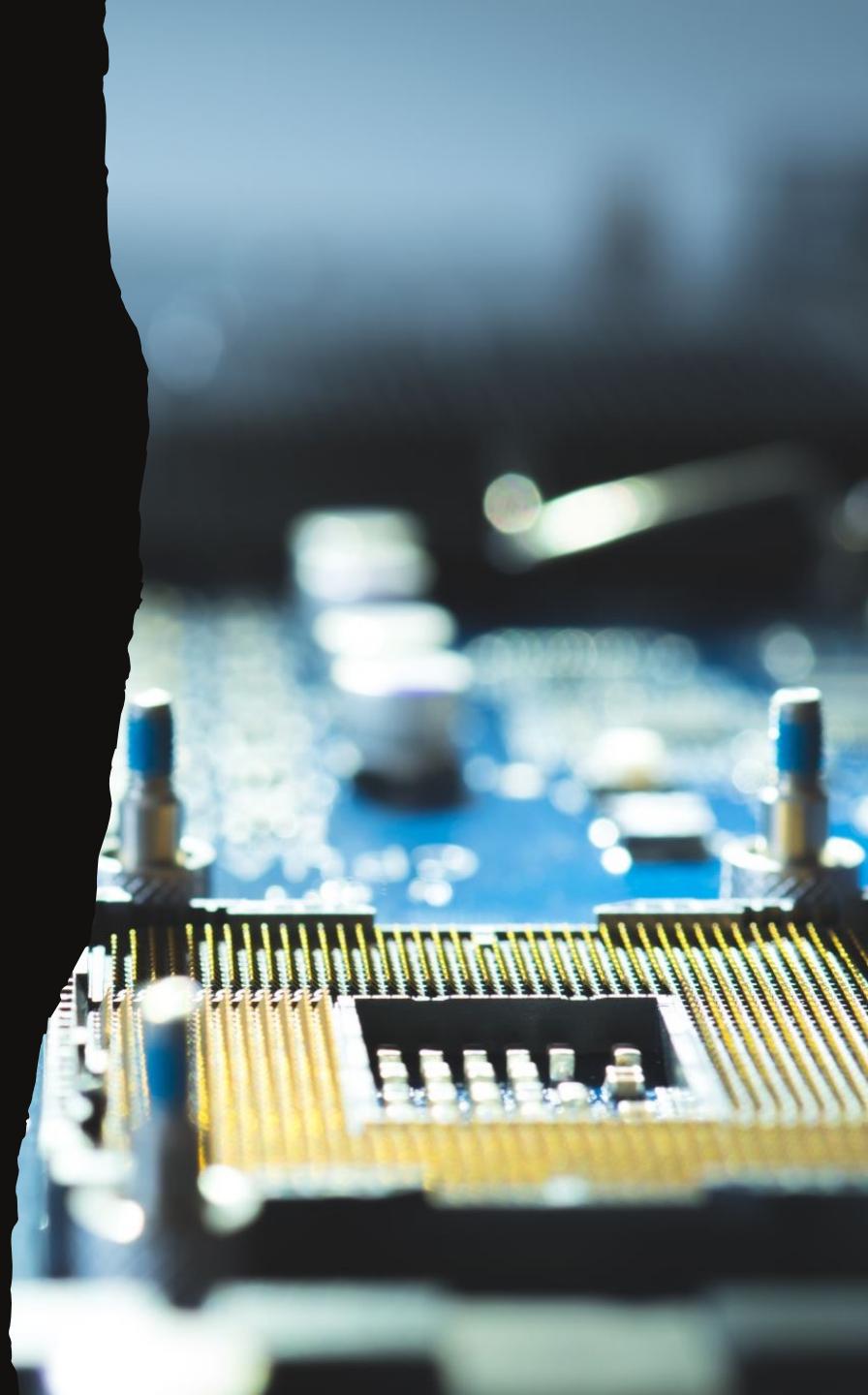


Education

- Boston University
 - Master of Science, Computer Science
- University of Wisconsin-Madison
 - Bachelor of Science, Computer Science
- Bellevue College
 - Associate of Arts and Sciences with Distinction, Computer Science

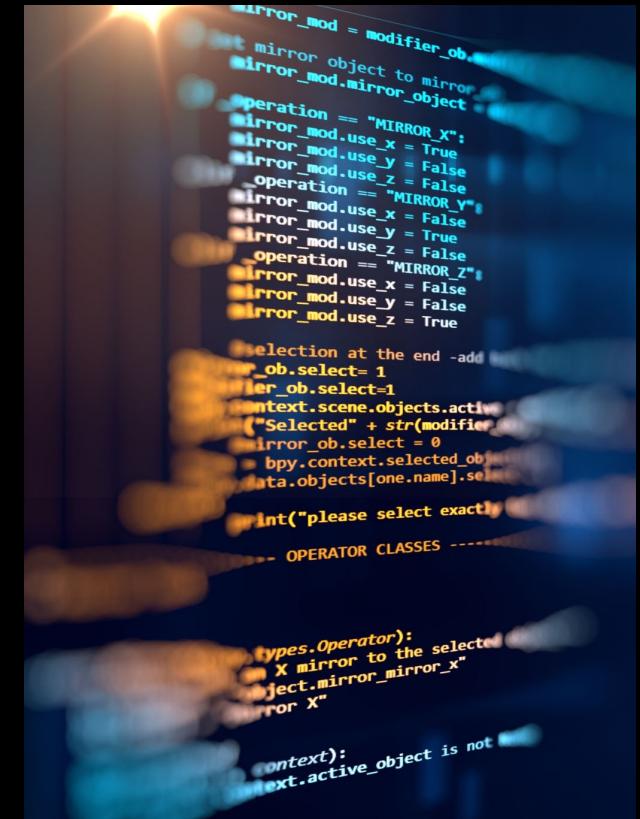
Software Engineer at Intel Taipei, 2021

- Cross-validation-testing Team in Wireless Connectivity Solution
- Implemented and Maintained Software Regression Automation Validation Tools
- Maintained automation Cloud-based Infrastructure Titan to monitor each RF shielded station.
 - Created automatic scripts and wrote documentations to resolve configuration issues.
 - Set up testing operating system by scripting languages and arranged RF shielded room environments.
- Intel Wireless Adapter Driver Validation
 - Delivered accurate develop team.
 - Differentiated issues affected by which develop teams and discussed on JIRA platform.
- Tested OSI Physical Layer of Wireless Adapter Driver (Bluetooth/WiFi) with Microwave Analyzer in WCS Lab
 - Ensured daily regression delivered expected performance with each dev commit.
 - Clarified physical issues not related daily regression results from 100+ PC stations located in Israel, India, Taiwan to code to driver performance.
 - Researched the Transmission and Reception performance with testing flows in HIT Manager and Keysight equipment.



Engineering Intern at Google Taipei, Summer 2023

- RF Software Team in Pixel Mobile Wireless
- Designed Python Script for merging C++ Configuration files, inputting to signaling equipment.
- Implemented infrastructure tools *Piper*, *google3*, *CLs* for designing Python Script, connecting input data for Pixel RF testing equipment.
- Utilized SQL querying functions from automation framework imported in *Spanner* database to generate combo key data structures.
- Created 2 Technical design docs for code reference.
- Presented 5G communication topic *Registration Procedure* with recorded videos.

A photograph of a person's hand pointing towards a computer monitor. The monitor displays a block of Python code. The code appears to be related to 3D modeling or rendering, specifically dealing with mirrors and object selection. It includes comments like "mirror_mod = modifier_obj", "mirror object to mirror", and "operation == "MIRROR X"". The background is dark, and the screen is brightly lit, creating a glow around the text.

```
mirror_mod = modifier_obj
# mirror object to mirror
mirror_mod.mirror_object

operation == "MIRROR X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
operation == "MIRROR Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
operation == "MIRROR Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

selection at the end -add
_ob.select= 1
ler_ob.select=1
ntext.scene.objects.active
("Selected" + str(modifier))
mirror_ob.select = 0
bpy.context.selected_objects
data.objects[one.name].sele
int("please select exactly one object")
-- OPERATOR CLASSES --
types.Operator):
    X mirror to the selected object.mirror_mirror_X"
    "mirror X"

(context):
    context.active_object is not None
```

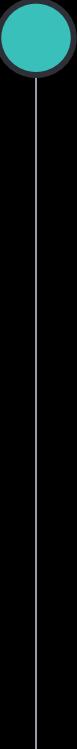
Social Links

- <https://www.linkedin.com/in/yiyang-lin/>
- <https://github.com/yiyanglin0102/>
- <https://www.facebook.com/yiyanglin0102/>
- <https://yiyanglin.com/>

Google Work



Intern Project



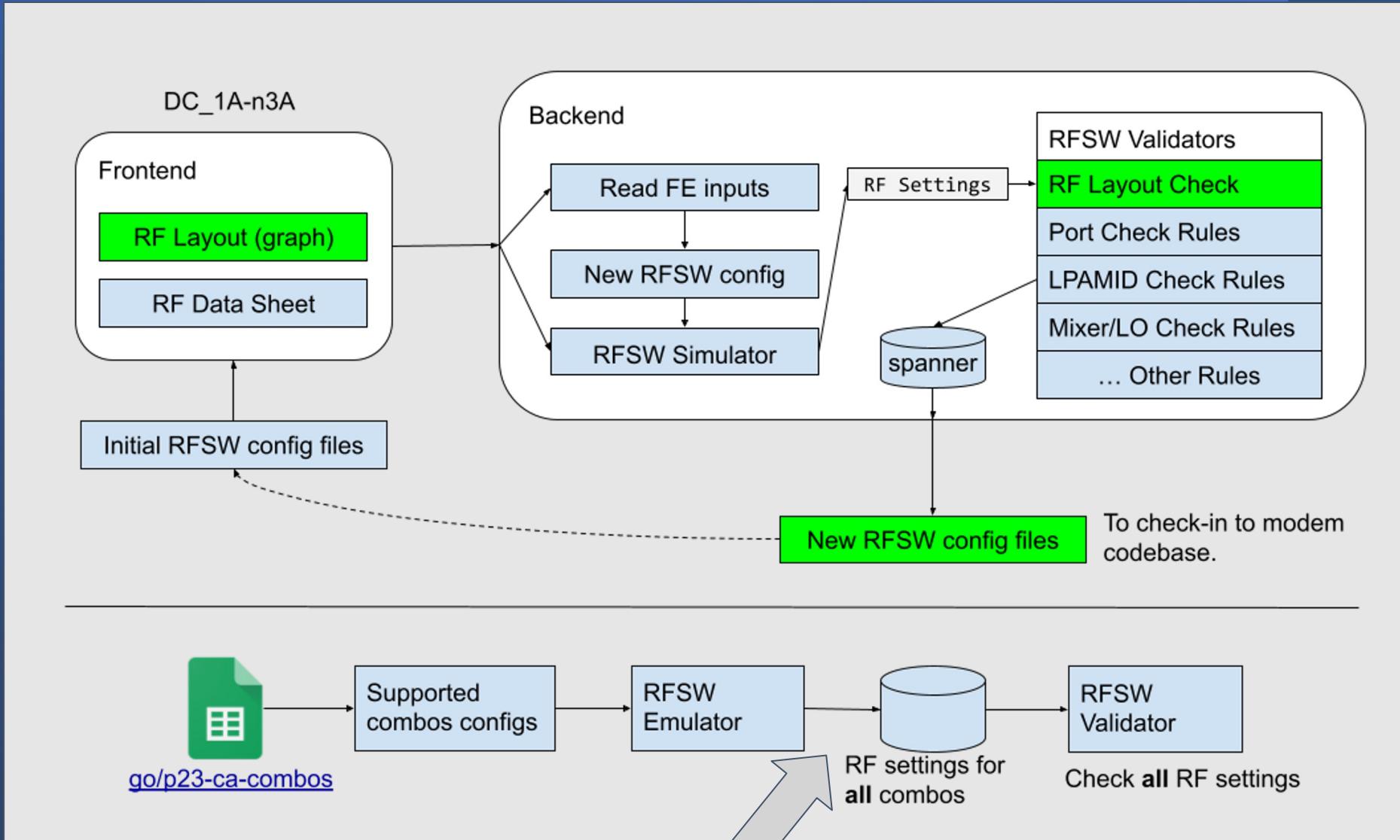
POR Testcase Script Generate

Outline

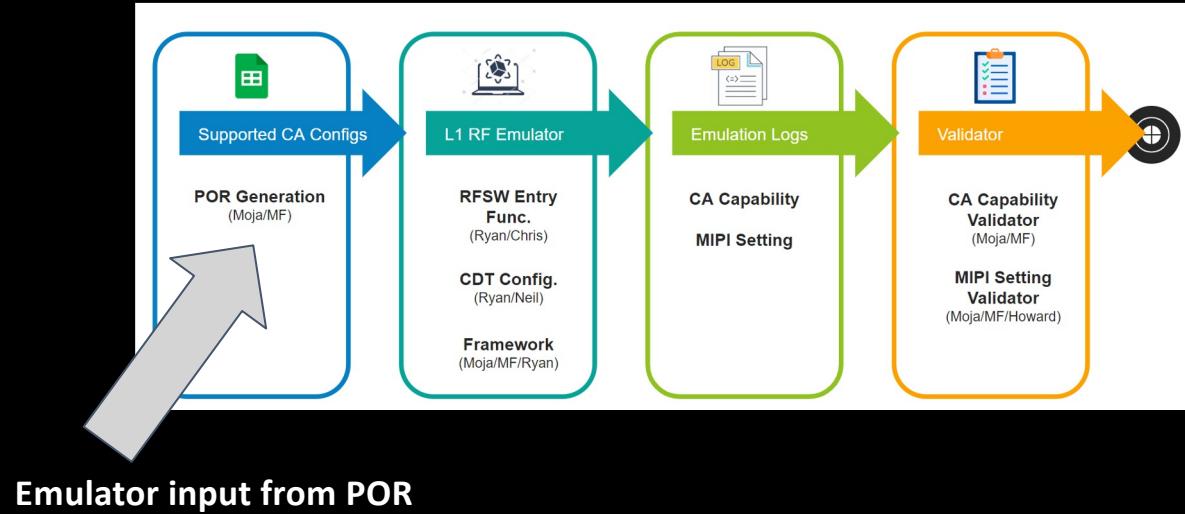
- 
- Background
 - Objective
 - Source Code
 - Code Build Process
 - Testcase Generation from POR for L1 RF Regression Code Review
 - Trace Code
 - RF Terminology

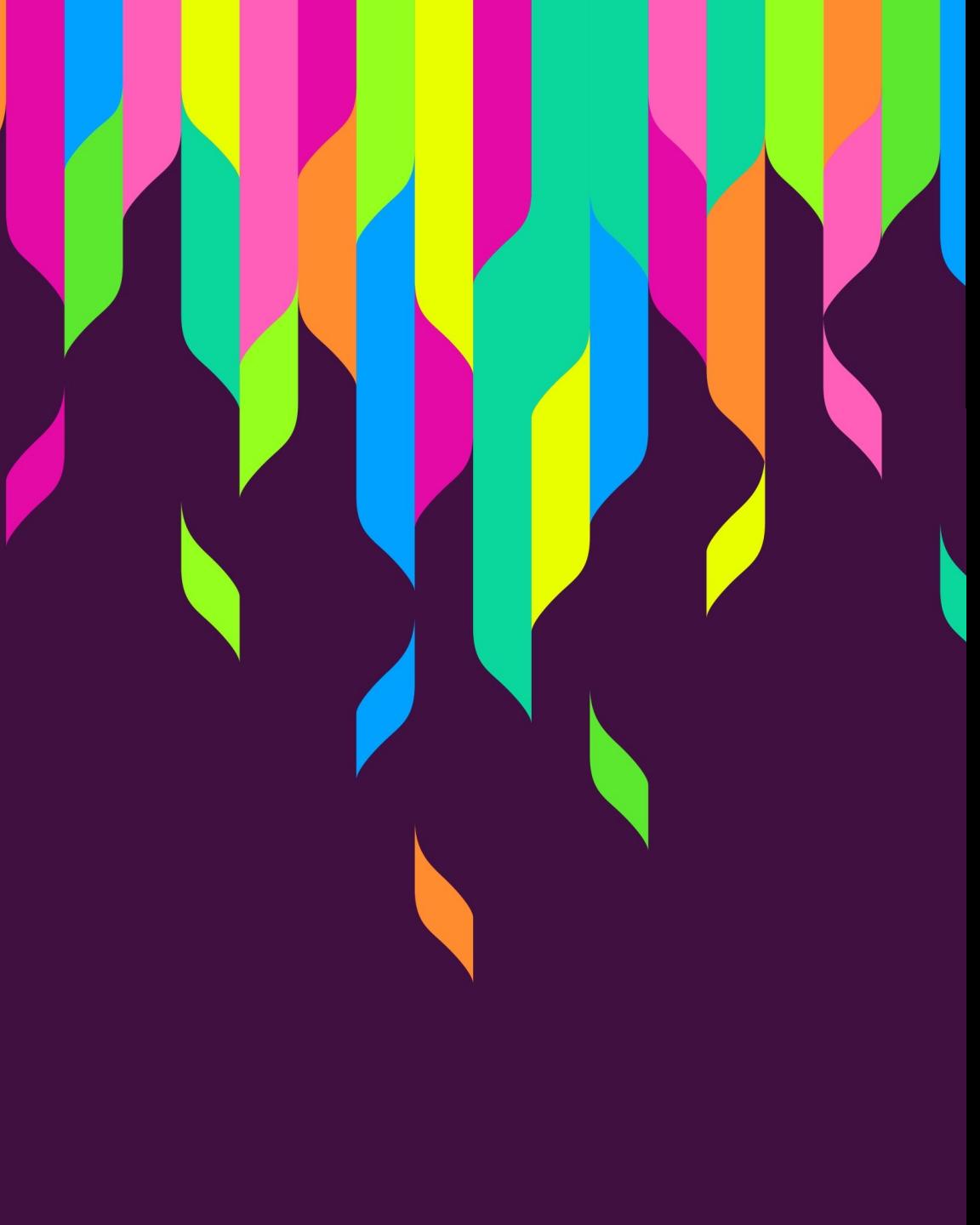
Background





Testcase Generation from POR for L1 RF Regression

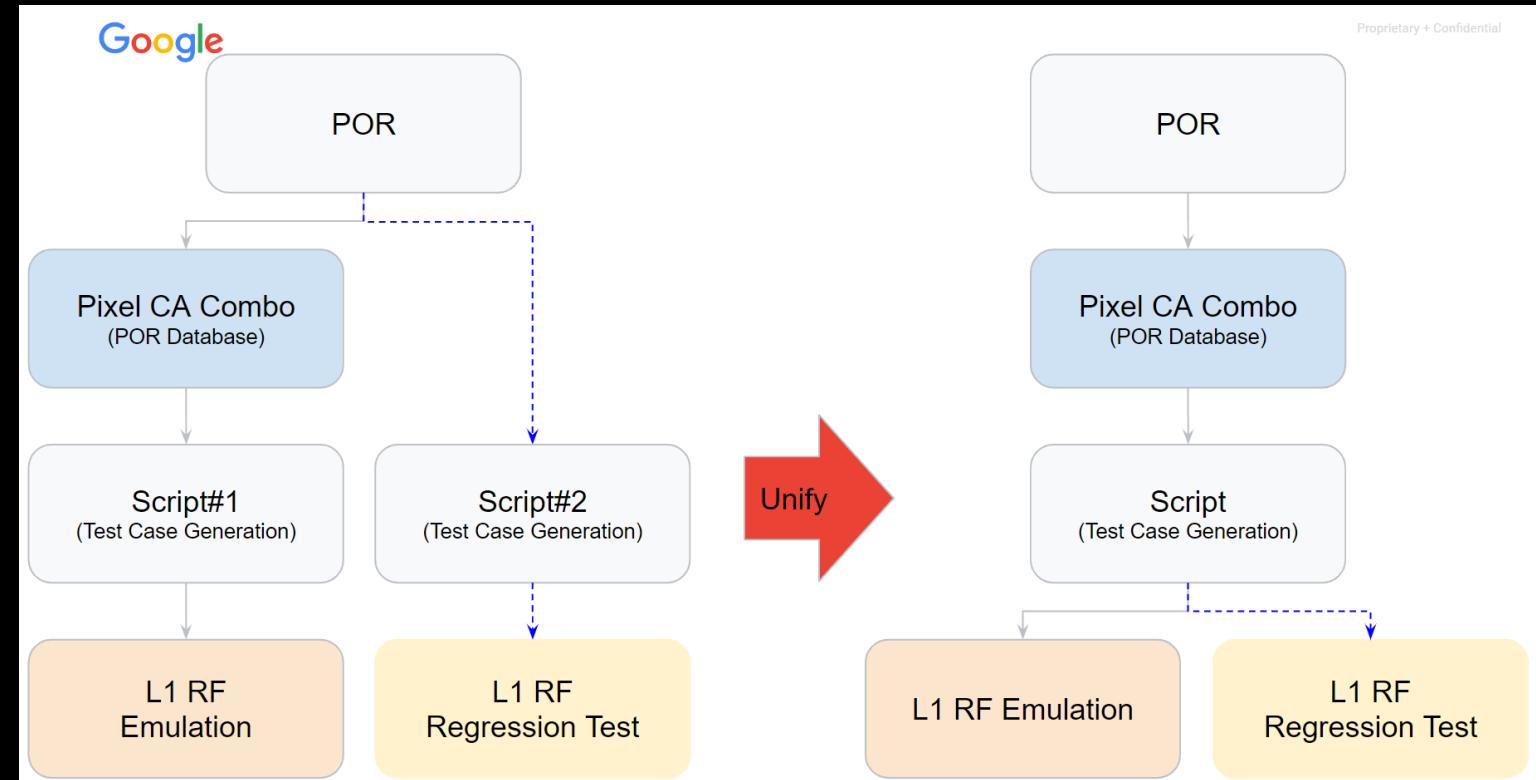




Objective

- Creating script for connecting POR to
- L1 RF Regression Tests
- By adding methods in
`rf_ca_config_generator_main.py`
- *Main*
- Merging L1 RF Emulation and L1 RF
Regression Test Scripts

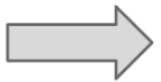
L1 RF Regression Input File Format



POR

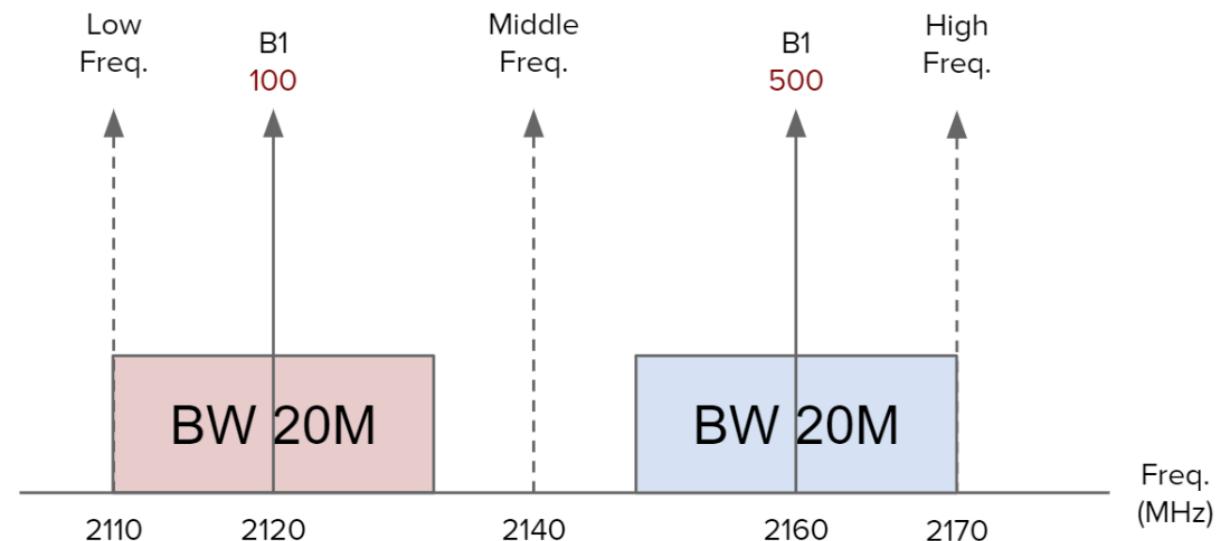
go/l10-ca-combos

US mmW SKU	
To see what is in the codebase (Should match!) see go/ca-combos-dash	
Total=13623	Default UL and MIMO config US mmW SKU
Configuration Name	DC_1A-1A-28A_n7A DC_28A_n7A(2-2-2-2)[[20]]



Test Case

28A-1A-1A_n7A#(28A-1A-1A,9310-100-500,20M-20M-20M,2-2-2);(n7A,531000,50M,2)



Format :

Combo_Name #

```
(  
    LTE_Band#1-.....-LTE_Band#n ,  
    EArfcn#1-.....EArfcn#n ,  
    BW#1-.....-BW#n ,  
    MIMO#1-.....-MIMO#n  
)  
;  
(  
    NR_Band#1-.....-NR_Band#n ,  
    NR-Arfcn#1-.....NR-Arfcn#n ,  
    BW#1-.....-BW#n ,  
    MIMO#1- .....-MIMO#n  
)
```

Generated L1 RF Regression Input File Format

- 8A - 3A_n41A # (8A-3A, 36525-1575, 10M-20M, 2-4);(n41A, 518598, 100M, 4)

Format :

Combo_Name #

(

LTE_Band#1-.....-LTE_Band#n ,

EArfcn#1-.....EArfcn#n ,

BW#1-.....-BW#n ,

MIMO#1-.....-MIMO#n

)

;

(

NR_Band#1-.....-NR_Band#n ,

NR-Arfcn#1-.....NR-Arfcn#n ,

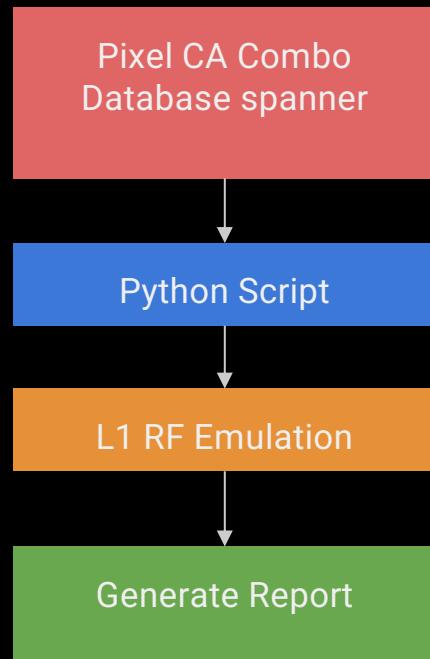
BW#1-.....-BW#n ,

MIMO#1--MIMO#n

)

Testcase Generation from POR for L1 RF Regression

Steps prior to generating report



POR

L10 NR & LTE combos (go/m23-ca-combos, go/l10-ca-combos) ★ ⓘ

File Edit View Insert Format Data Tools Extensions Help

Comment only

A1

	A	B	C	D	E	F
	CL		To see what is in the database (Should match!) see ca_configs-dash	US mmW SKU	NA sub6 SKU	
			Total=13623			
Search keyword	Date added	POR in L10	Configuration Name	Default UL and MIMO config US mmW SKU	Default UL and MIMO config NA sub6 SKU	
DCx1AnAx	03-16-2020	Y	DC_1A_n8A	DC_1A_n8A(2-2)([10])	DC_1A_n8A(2-2)([10])	
DCx1An7Ax	03-16-2020	Y	DC_1A_n1A	Not supported	Not supported	
DCx1An3Ax	03-16-2020	Y	DC_1A_n3A	Not supported	Not supported	
DCx1An3An7B8Ax	03-16-2020	Y	DC_1A_n3A_n7BA	DC_1A_n7BA(2-2-4)([20],[100])	DC_1A_n7BA(2-2-4)([20],[100])	
DCx1An2B8Ax	03-16-2020	Y	DC_1A_n2BA	DC_1A_n2BA(2-2)([10])	DC_1A_n2BA(2-2)([10])	

Pixel CA Combos

data.corp.google.com/sites/mhb5af6f3ggp/home/ Pixel CA Combos (go/ca-combos-dash)

N Combos on Google3 CL CL Combo Diff

Add filter

CL_ID	Combo Configs	Carrier Summary
513703904		

CL TI

Release Version Project

RELEASE_M23_TA C10P10: go/p22-ca-combos-mar23

RELEASE_QPR_MAR_2023 C10P10: go/p22-ca-combos-mar23

RELEASE_QPR_DEC_2022 C10P10: go/p22-ca-combos-dec22

RELEASE_P22_TA C10P10: go/p22-ca-combos

Project Carrier ComboName ↑

L10 VZW CA_13A-48A

ca_configs.cpp

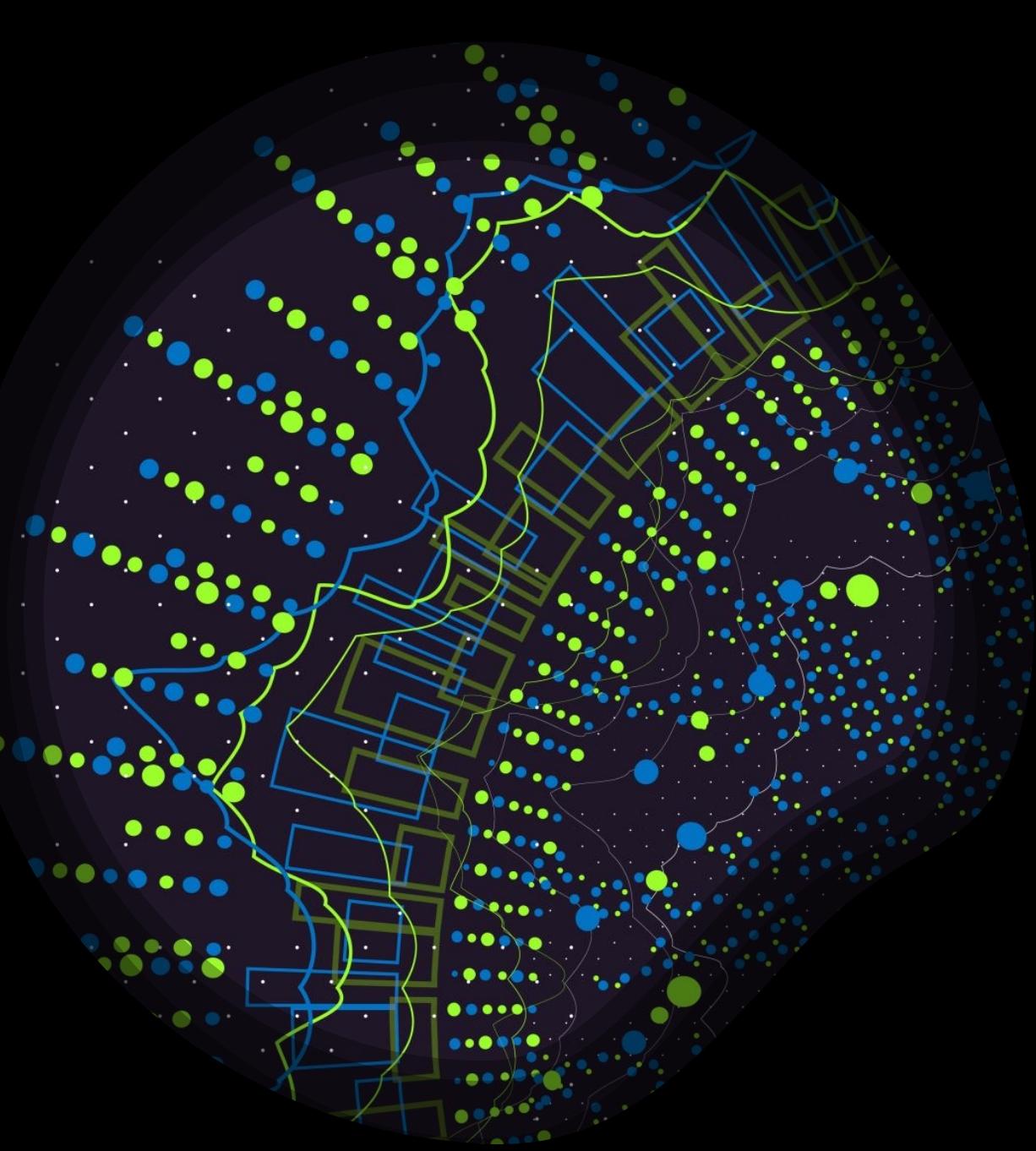
```
extern "C" {
#include "ca_configs.h"
}

const CaConfig_t ca_12_25_p12 = {
    .name = "ca_12_25_p12",
    .band_cnt = 2,
    .bandConfigList = {
        {RF_BAND_12, false, true},
        {RF_BAND_25, false, false},
    },
};
```

l10_mmw.cpp

```
#include <vector>
extern "C" {
#include "ca_configs.h"
}

std::vector<const CaConfig_t*> l10_sub6_row_ca_configs {
    &ca_12_2_p12,
    &ca_12_2_4_p12,
    &ca_12_2_66_p12,
    &ca_12_2_7_p12,
    &ca_12_2_p12,
    &ca_12_4_7_p12,
    &ca_12_4_p12,
    &ca_12_66_p12,
    &ca_12_7_66_p12,
    &ca_12_7f_p12,
    &ca_12_p12,
    &ca_17_2_p17,
```



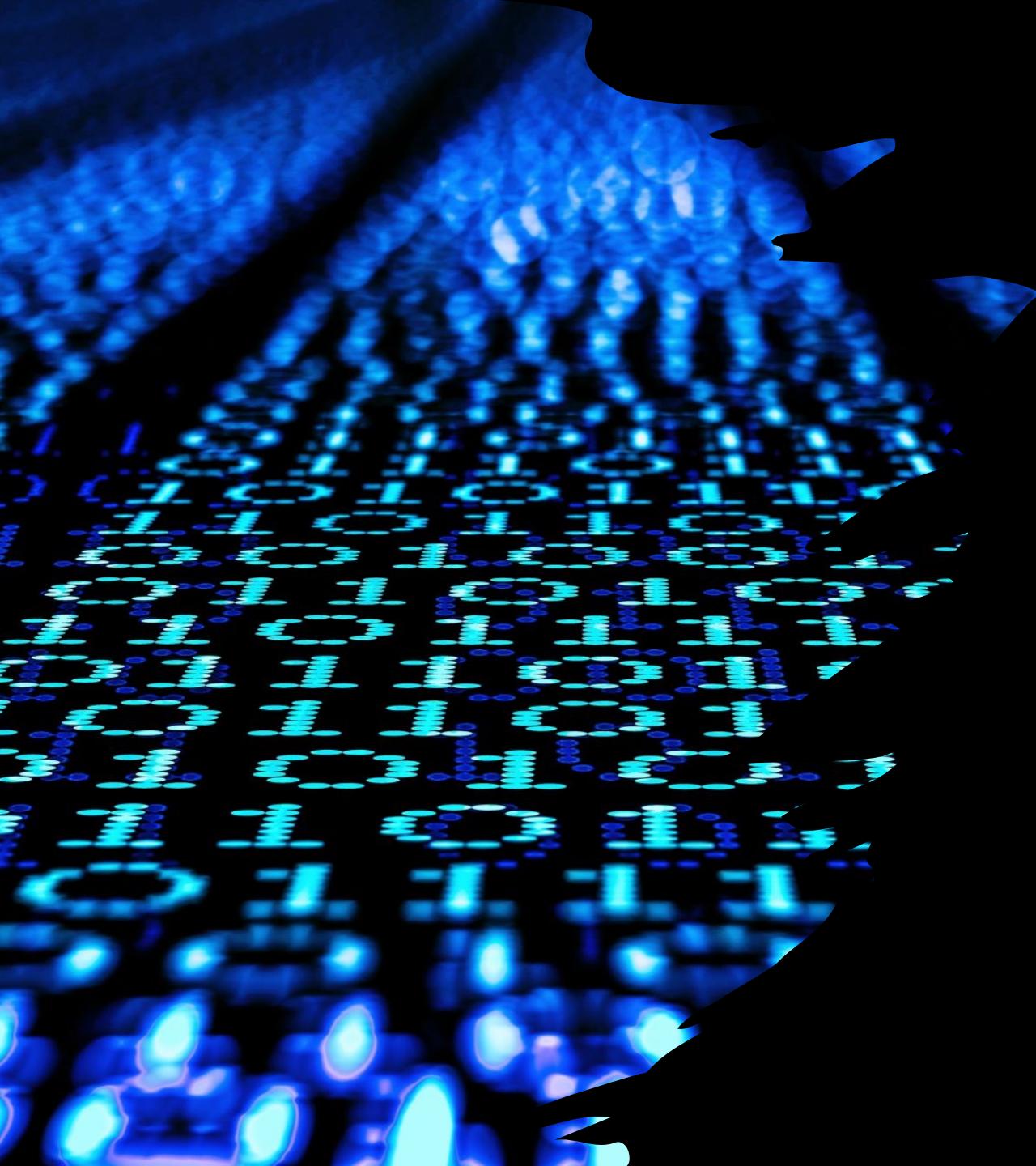
Infrastructure and Tools

- Editor
 - Cider-V
- *Lab go/dwf*
- Get access
 - gcert ssh
 - Corp Normal
- Cloudtop
 - Chrome Remote Desktop
- Database
 - Spanner
- Blaze
- google3
 - *Piper*
 - */google3/wireless/android/pixel/modem*
 - */ca_combo_gen/ca_combo_gen/rf_ca_config_gen/*



Build Process

- Understand Loading Dependencies from google3
 - proto/ .proto files
 - Database - Spanner
- Blaze
 - Command under the directory
 - */google/src/cloud/username/workspace/google3/wireless/android/pixel/modem*
 - */ca_combo_gen/ca_combo_gen/rf_ca_config_gen*
 - *blaze run rf_ca_config_generator_main*
- BUILD File Structure



BUILD File

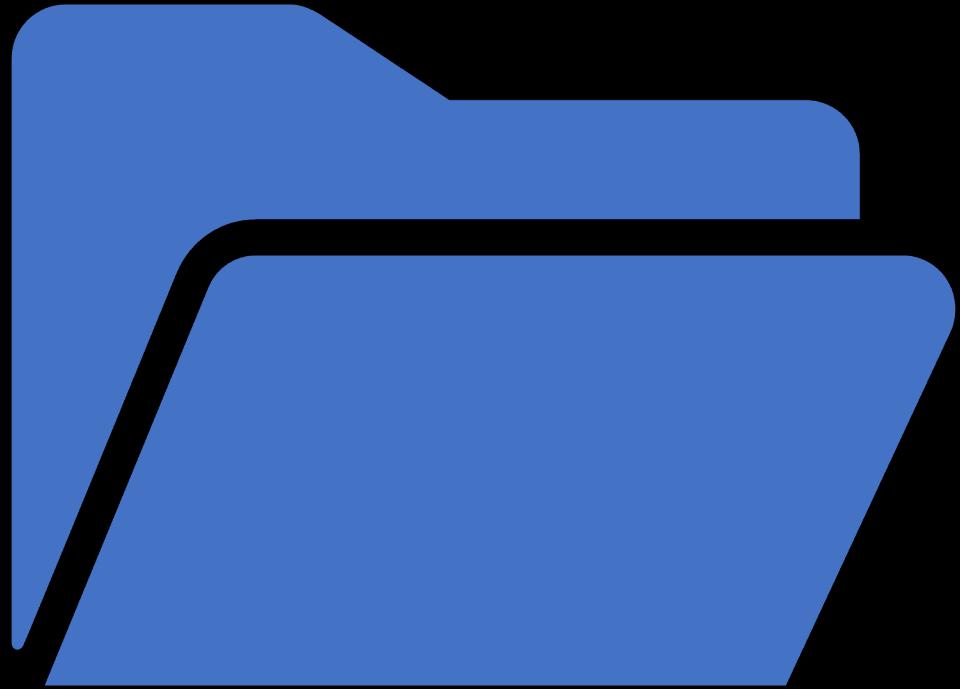
- Containing the rules that *Blaze* uses to build that directory and its children into the package.
- A list of targets, their sources, and their dependencies describing rules.
- Similar to *Makefile* in Unix / Linux
- Import dependencies from Infrastructure *proto*, *spanner* ... etc.
- Main entry point for Python build configuration workflow

```
rf_ca_config_gen
BUILD
rf_ca_config_generator.py
rf_ca_config_generator_main.py
rf_ca_config_generator_test.py

1 load("//devtools/python/blaze:pytype.bzl", "pytype_strict_binary", "pytype_strict_contrib_test", "pytype_strict_library")
2
3 pytype_strict_library(
4     name = "rf_ca_config_generator",
5     srcs = ["rf_ca_config_generator.py"],
6     deps = [
7         "//devtools/production/pyspanner",
8         "//net/rpc2/contrib/smartservice/python:smartservice_util",
9         "//pyglib:gfile",
10        "//wireless/android/pixel/modem/modem_config/database:modem_config_db",
11        "//wireless/android/pixel/modem/modem_config/proto:checked_combo_py_pb2",
12        "//wireless/android/pixel/modem/modem_config/proto:combo_configuration_py_pb2",
13        "//wireless/android/pixel/modem/modem_config/proto:combo_py_pb2",
14        "//wireless/android/pixel/modem/modem_config/proto:pixel_project_py_pb2",
15        "//wireless/android/pixel/modem/modem_config/proto:rpc_py_pb2",
16        "//wireless/android/pixel/modem/modem_config/proto:specs_requirements_py_pb2",
17    ],
18 )
19
20 pytype_strict_contrib_test(
21     name = "rf_ca_config_generator_test",
22     srcs = ["rf_ca_config_generator_test.py"],
23     deps = [
24         ":rf_ca_config_generator",
25         "//devtools/production/pyspanner",
26         "//net/rpc2/contrib/smartservice/python:smartservice_util",
27         "//pyglib:gfile",
28         "//testing/pybase",
29         "//testing/pybase:parameterized",
30         "//wireless/android/pixel/modem/modem_config/database:modem_config_db",
31         "//wireless/android/pixel/modem/modem_config/proto:checked_combo_py_pb2",
32         "//wireless/android/pixel/modem/modem_config/proto:combo_configuration_py_pb2",
33         "//wireless/android/pixel/modem/modem_config/proto:combo_py_pb2",
34         "//wireless/android/pixel/modem/modem_config/proto:pixel_project_py_pb2",
35         "//wireless/android/pixel/modem/modem_config/proto:rpc_py_pb2",
36         "//wireless/android/pixel/modem/modem_config/proto:specs_requirements_py_pb2",
37    ],
38 )
39
40 pytype_strict_binary(
41     name = "rf_ca_config_generator_main",
42     srcs = ["rf_ca_config_generator_main.py"],
43     deps = [
44         ":rf_ca_config_generator",
45         "//devtools/production/pyspanner",
46         "//file/colossus/public:cns",
47         "//pyglib:gfile",
48         "//third_party/py/absl:app",
49    ],
50 )
```

Trace Codes

- 
- `rf_ca_config_generator.py`
 - **main file called by**
 - *rf_ca_config_generator_main*
 - `rf_ca_config_generator_main.py`
 - `rf_ca_config_generator_test.py`



rf_ca_config_generator_main.py

- Main entry point for POR generation script
- Define the schema using the Protocol Buffers language (.proto file) in Enum struct
- Hard-coded *PixelProjectName*, *SkuName*, *DeviceVersion*, *PixelRelease* information to input into script in self-defined structure proto

rf_ca_config_generator.py

- Outputting *ca_configs.h* , *ca_configs.cpp* , *{project}_{sku}.cpp* files
- Implementing conversion CA Config Naming rules from

[go/pmw-ti-rfsw-por-gen-dd](http://pmw-ti-rfsw-por-gen-dd)

rf_ca_config_gen
out
ak3_jp.cpp
ak3_mmw.cpp
ak3_sub6_na.cpp
ak3_sub6_row.cpp
c10_mmw.cpp
c10_sub6.cpp
ca_configs.cpp
ca_configs.h
f10_mmw.cpp
hk3_jp.cpp
hk3_mmw.cpp
hk3_sub6.cpp
l10_mmw.cpp
l10_sub6_na.cpp
l10_sub6_row.cpp
p10_mmw.cpp
p10_sub6.cpp
sb3_jp.cpp
sb3_mmw.cpp
sb3_sub6.cpp

→ output file

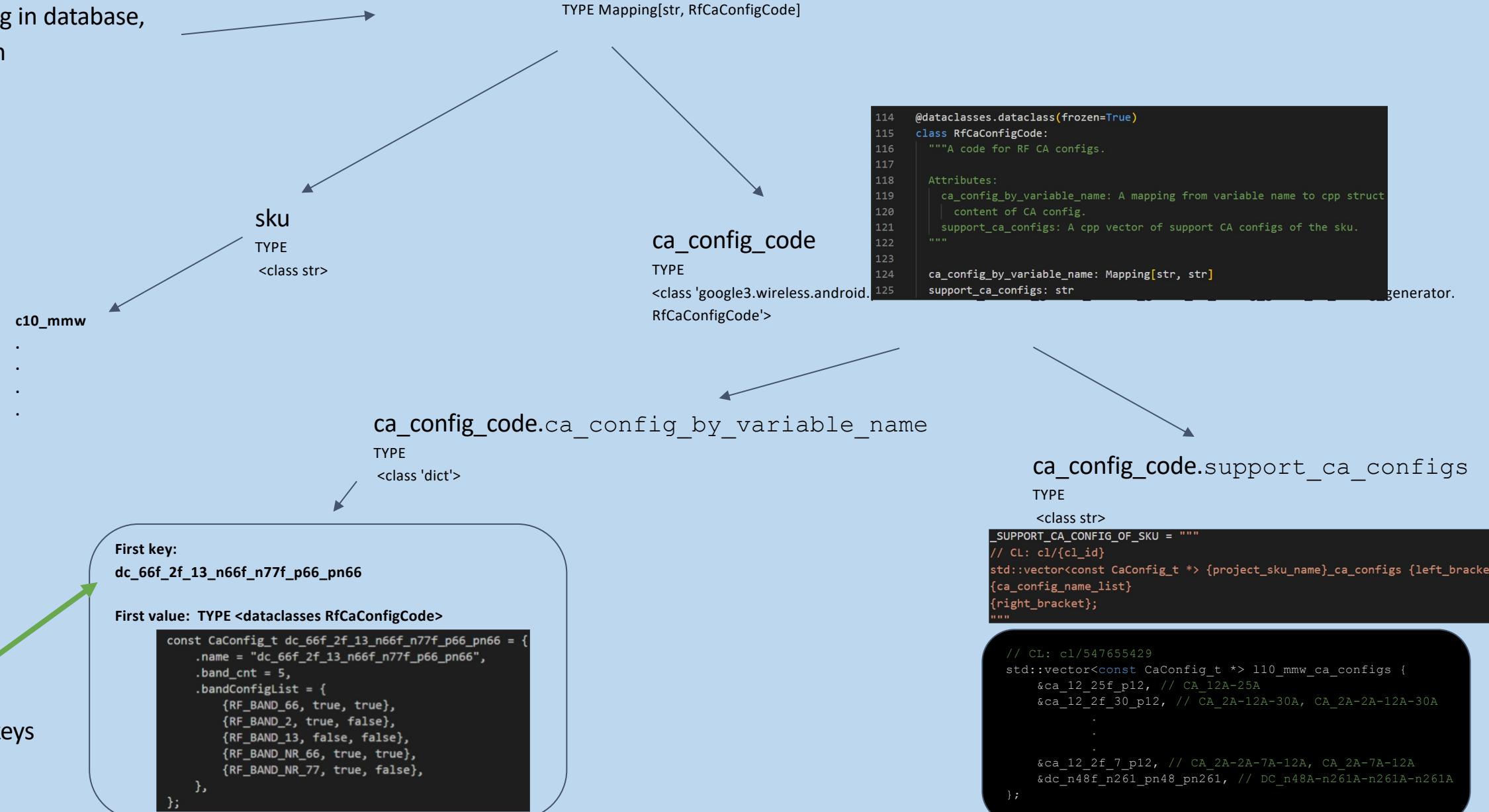
2. ca_configs.h

```
with gfile.Open(os.path.join(device_dir, 'ca_configs.h'), 'w') as f:  
    f.write(_CONFIGS_H_HEADER)  
    for name in sorted(total_ca_configs_by_variable_name):  
        f.write(f'extern const CaConfig_t {name};\n')  
    f.write(_CONFIGS_H_TAIL)
```

```
def generate_ca_configs(  
    self,  
    device_name: str,  
    projects: Collection[pixel_project_pb2.PixelProjectName],  
    output_dir: str,  
) -> None:  
    """Generates CA configs code.  
  
    It generates total ca configs c struct code and support ca configs per SKU.  
  
    Args:  
        device_name: The device name.  
        projects: The projects of the device.  
        output_dir: The output directory for generated files.  
    """  
    device_dir = os.path.join(output_dir, device_name)  
    gfile.MakeDirs(device_dir)  
  
    support_ca_configs_code_by_sku = (  
        self.generate_support_ca_configs_code_of_sku_from_projects(projects)  
    )  
  
    # Collects all ca configs from all SKUs, we need to put them in a cpp file.  
    total_ca_configs_by_variable_name: dict[str, str] = {}  
    for sku, ca_config_code in support_ca_configs_code_by_sku.items():  
        total_ca_configs_by_variable_name.update(  
            {ca_config_code.ca_config_by_variable_name:  
                []})  
        # with gfile.Open(os.path.join(device_dir, f'{sku}.cpp'), 'w') as f:  
        #     f.write(_SUPPORT_C_HEADER)  
        #     f.write(ca_config_code.support_ca_configs)  
  
    with gfile.Open(os.path.join(device_dir, 'ca_configs.h'), 'w') as f:  
        f.write(_CONFIGS_H_HEADER)  
        for name in sorted(total_ca_configs_by_variable_name):  
            f.write(f'extern const CaConfig_t {name};\n')  
        f.write(_CONFIGS_H_TAIL)  
  
    # with gfile.Open(os.path.join(device_dir, 'ca_configs.cpp'), 'w') as f:  
    #     f.write(_CONFIGS_C_HEADER)  
    #     for name in sorted(total_ca_configs_by_variable_name):  
    #         f.write(total_ca_configs_by_variable_name[name])  
  
    print(  
        f'Use the following command to get generated files for {device_name}.'  
)  
    print(f'fileutil cp {device_dir}/* .')
```

```
#ifndef __CA_CONFIGS_H__  
#define __CA_CONFIGS_H__  
extern "C" {  
#include "ca_config.h"  
}  
extern const CaConfig_t ca_12_25_p12;  
extern const CaConfig_t ca_12_25f_p12;  
extern const CaConfig_t ca_12_2_4_p12;  
extern const CaConfig_t ca_12_2_66_p12;  
extern const CaConfig_t ca_12_2_7_p12;  
extern const CaConfig_t ca_12_2_p12;  
extern const CaConfig_t ca_12_2f_30_p12;  
extern const CaConfig_t ca_12_2f_30f_66f_p12;  
extern const CaConfig_t ca_12_2f_30f_p12;  
extern const CaConfig_t ca_12_2f_4f_30f_p12;  
extern const CaConfig_t ca_12_2f_4f_7f_p12;  
extern const CaConfig_t ca_12_2f_4f_p12;  
extern const CaConfig_t ca_12_2f_66f_p12;  
extern const CaConfig_t ca_12_2f_7_p12;  
extern const CaConfig_t ca_12_2f_7f_66f_p12;  
extern const CaConfig_t ca_12_2f_7f_p12;  
extern const CaConfig_t ca_12_2f_p12;  
:  
:  
:  
:  
extern const CaConfig_t dc_n77f_n261_n2f_pn77_pn261;  
extern const CaConfig_t dc_n77f_n261_n5_pn77_pn261;  
extern const CaConfig_t dc_n77f_n261_n66f_pn77_pn261;  
extern const CaConfig_t dc_n77f_n261_pn77_pn261;  
extern const CaConfig_t dc_n78f_n258_pn78_pn258;  
#endif
```

After searching in database,
the Result is in



3. ca_configs.cpp

```
with gfile.Open(os.path.join(device_dir, 'ca_configs.cpp'), 'w') as f:
    f.write(_CONFIGS_C_HEADER)
    for name in sorted(total_ca_configs_by_variable_name):
        f.write(total_ca_configs_by_variable_name[name])
```

```
def generate_ca_configs(
    self,
    device_name: str,
    projects: Collection[pixel_project_pb2.PixelProjectName],
    output_dir: str,
) -> None:
    """Generates CA configs code.

    It generates total ca configs c struct code and support ca configs per SKU.

    Args:
        device_name: The device name.
        projects: The projects of the device.
        output_dir: The output directory for generated files.
    """
    device_dir = os.path.join(output_dir, device_name)
    gfile.MakeDirs(device_dir)

    support_ca_configs_code_by_sku = (
        self.generate_support_ca_configs_code_of_sku_from_projects(projects)
    )
    # Collects all ca configs from all SKUs, we need to put them in a cpp file.
    total_ca_configs_by_variable_name: dict[str, str] = {}
    for sku, ca_config_code in support_ca_configs_code_by_sku.items():
        total_ca_configs_by_variable_name.update(
            ca_config_code.ca_config_by_variable_name
        )
    # with gfile.Open(os.path.join(device_dir, f'{sku}.cpp'), 'w') as f:
    #     f.write(_SUPPORT_C_HEADER)
    #     f.write(ca_config_code.support_ca_configs)

    # with gfile.Open(os.path.join(device_dir, 'ca_configs.h'), 'w') as f:
    #     f.write(_CONFIGS_H_HEADER)
    #     for name in sorted(total_ca_configs_by_variable_name):
    #         f.write(f'extern const CaConfig_t {name};\n')
    #     f.write(_CONFIGS_H_TAIL)

    with gfile.Open(os.path.join(device_dir, 'ca_configs.cpp'), 'w') as f:
        f.write(_CONFIGS_C_HEADER)
        for name in sorted(total_ca_configs_by_variable_name):
            f.write(total_ca_configs_by_variable_name[name])

    print(
        f'Use the following command to get generated files for {device_name}.'
    )
    print(f'fileutil cp {device_dir}/* .')
```

→ output file

```
extern "C" {
#include "ca_configs.h"
}

const CaConfig_t ca_12_25_p12 = {
    .name = "ca_12_25_p12",
    .band_cnt = 2,
    .bandConfigList = {
        {RF_BAND_12, false, true},
        {RF_BAND_25, false, false},
    },
};

const CaConfig_t ca_12_25f_p12 = {
    .name = "ca_12_25f_p12",
    .band_cnt = 2,
    .bandConfigList = {
        {RF_BAND_12, false, true},
        {RF_BAND_25, true, false},
    },
};

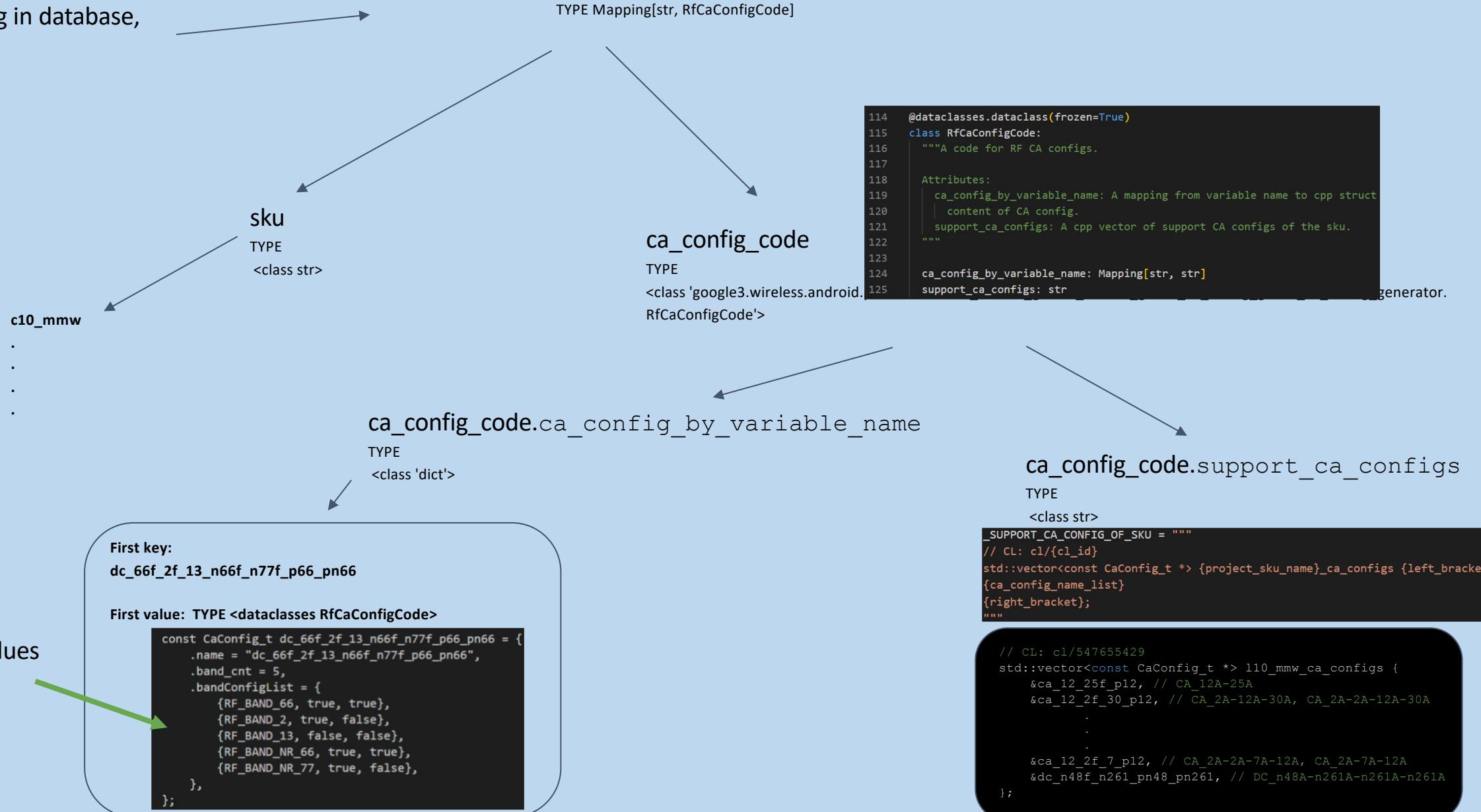
const CaConfig_t ca_12_2_4_p12 = {
    .name = "ca_12_2_4_p12",
    .band_cnt = 3,
    .bandConfigList = {
        {RF_BAND_12, false, true},
        {RF_BAND_2, false, false},
        {RF_BAND_4, false, false},
    },
};

const CaConfig_t ca_12_2_66_p12 = {
    .name = "ca_12_2_66_p12",
    .band_cnt = 3,
    .bandConfigList = {
        {RF_BAND_12, false, true},
        {RF_BAND_2, false, false},
    },
};

const CaConfig_t dc_n77f_n261_pn77_pn261 = {
    .name = "dc_n77f_n261_pn77_pn261",
    .band_cnt = 2,
    .bandConfigList = {
        {RF_BAND_NR_77, true, true},
        {RF_BAND_NR_257_261, false, true},
    },
};

const CaConfig_t dc_n78f_n258_pn78_pn258 = {
    .name = "dc_n78f_n258_pn78_pn258",
    .band_cnt = 2,
    .bandConfigList = {
        {RF_BAND_NR_78, true, true},
        {RF_BAND_NR_258, false, true},
    },
};
```

After searching in database,
the Result is in



fileutil

- [go/fileutil](#)
- [go/copy-tools](#)
- `fileutil cp` is a command-line tool for copying a few files from place A to place B. It behaves like the UNIX cp command with the Google infrastructure and allows you to copy between corp and production, in both directions. For example, use `fileutil cp` to copy data out of Colossus to your local disk.
- `fileutil cp <local file> <cns address>`
- eg.

fileutil cp /cns/ed-d/home/**username**/ttl=120d/rf_ca_configs/20230713061533/m23/* ./local_dir

Copy files and directories from one location to another

The local directory in gLinux/cloudtop

Data will be persisted to CNS, to a shared directory for all tasks.

time-to-live (TTL) value. In this case, the TTL is 120 days. TTL is measured against the file or directory mtime (last modified time). When the TTL is exceeded, the file (or directory) is irrevocably deleted.

```
liniyang@liniyang:/google/src/cloud/liniyang/Test2/google3/wireless/android/pixel/modem/ca_combo_gen/ca_combo_gen$ fileutil ls /cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230629065658  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230629072102  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230629082525  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704063800  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704063901  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704064240  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704064711  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704065134  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704065529  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704070210  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704071126  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704071757  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704072045  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704072338  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704072605  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704073029  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704073205  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704073508  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704073637  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704073757  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704073920  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704121244  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704121419  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230704122838  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230705013143  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230705015042  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230705052725  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230705053019  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230705053439  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230705053626  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230705054128  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230705120011  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230706001004  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230706011126  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230706082106  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230706083434  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230707081421  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230707085310  
/cns/ed-d/home/liniyang/ttl=120d/rf_ca_configs/20230710095517
```

Colossus

- [go/colossus-overview](#)
- /cns/ed-d/
 - Providing a durable (but not permanent, because clusters turn down) repository for large amounts of unstructured data, such as videos.
 - Providing temporary storage of data in large workflow systems, for example log files.

Database Spanner

- Google's globally distributed database
- [go/spanner-codelab-python](https://github.com/GoogleCloudPlatform/python-docs-samples/tree/master/spanner/codelab)

Source: *Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis non erat sem*

The image shows a developer's workspace with two main panels: a code editor on the left and a Data Marketplace interface on the right.

Code Editor (Left Panel):

- The title bar shows "modem_config_db.py x".
- The code editor displays Python code for a database module, specifically for handling modem configurations. It includes imports from `google3` and `pyspanner`, and defines several constants and tables related to CaComboConfigs.
- A white arrow points from the line `_PENDING_CA_COMBO_CONFIGS_TABLE = 'PendingSupportedCaComboConfigs'" in the code editor towards the "Schema" tab in the Data Marketplace.
- The status bar at the bottom shows "Test2" and other build-related information.

Data Marketplace (Right Panel):

- The title bar shows "Data Marketplace" and "spanner.global.pixel-modem-tools-modem-config-spanner.prod".
- The "Schema" tab is selected, displaying a list of tables and their definitions:

 - CaComboMetaData
 - CaComboStringRepresentation
 - CarrierFeatureConfig
 - CarrierMetadata
 - FeatureMetadata
 - LpamidCheckerResults
 - LpamidCheckerTestCases
 - PendingCaComboMetaData
 - PendingSupportedCaComboConfigs
 - QueryComboGraphs
 - RegisterValueMaps
 - SupersetComboStats
 - SupportedCaComboConfigs
 - VerifiedCaComboConfigs
 - WillitUpdateStub

- The "Table Description" section indicates the table supports CaComboConfigs with db path /span/global/pixel-modem-tools-modem-config-spanner:prod.
- The "Query Builder" tab shows a query:

```
1 SET RequestOptions.requested_enable_feature=LOAD_MISSING_SPANNER_TYPES_FROM_GLOBAL_PROTO_DB;
2 SET RequestOptions.requested_disable_feature=ENABLE_MAX_TOPOLOGY_DISTANCE_FILTER; # For possible
f1:DATA_LOCATION_ERROR: http://go/fierrror/max_topology_distance_data_location_error.md
3 SELECT *
4 FROM `/span/global/pixel-modem-tools-modem-config-spanner:prod`.SupportedCaComboConfigs AS t
5 LIMIT 10;
```

```
399     with gfile.Open(os.path.join(device_dir, 'ca_configs.h'), 'w') as f:
400         f.write(_CONFIGS_H_HEADER)
401         for name in sorted(total_ca_configs_by_variable_name):
402             f.write(f'extern const CaConfig_t {name};\n')
403         f.write(_CONFIGS_H_TAIL)
404
405     with gfile.Open(os.path.join(device_dir, 'ca_configs.cpp'), 'w') as f:
406         f.write(_CONFIGS_C_HEADER)
407         for name in sorted(total_ca_configs_by_variable_name):
408             f.write(total_ca_configs_by_variable_name[name])
409
410     print(
411         f'Use the following command to get generated files for {device_name}.'
412     )
413     print(f'fileutil cp {device_dir}/* .')
414
415     def generate_all_ca_configs(
416         self,
417         output_dir: str,
418     ) -> None:
419         """Generates CA configs code for all projects.
420
```

```
_config_generator_main.py rf_ca_config_generator.py 4 modem_config_db.py X rf_ca_config_generator_test.py
je3 > wireless > android > pixel > modem > modem_config > database > modem_config_db.py > ModemConfigDB > get_unique_project_names
960     if version_id is None:
961         return
962     for row in self._spanner_db.BindAndQuery(
963         _QueryUniqueProjectSkuComboKeys, [project, sku, version_id]
964     ):
965         yield row[0]
966
967     def get_unique_project_names(
968         self,
969     ) -> Iterator['pixel_project_pb2.PixelProjectName']:
970         """Lists the unique project names."""
971         for row in self._spanner_db.Query(_QueryUniqueProjectNames):
972             yield row[0]
973
```

```
def get_unique_project_names(
    self,
) -> Iterator['pixel_project_pb2.PixelProjectName']:
    """Lists the unique project names."""
    for row in self._spanner_db.Query(_QueryUniqueProjectNames):
        yield row[0]
```

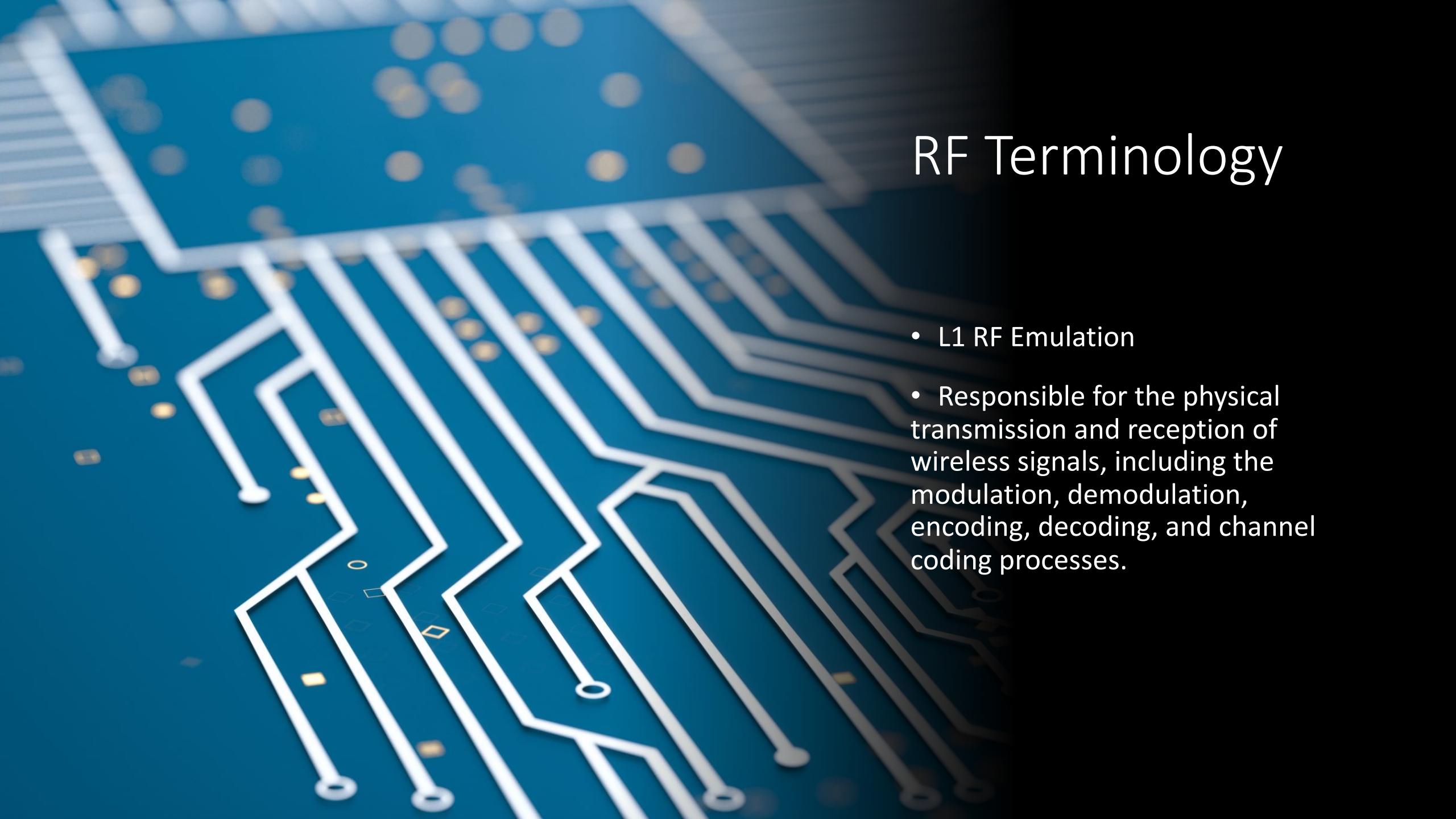
modem_config_db.py X

google3 > wireless > android > pixel > modem > modem_config > database > modem_config_db.py > _QueryUniqueProjectNames

```
319  
320     class _QueryUniqueProjectNames(pyspanner.Query):  
321         """# Queries the unique project names.  
322  
323             SELECT DISTINCT  
324                 t.ProjectName  
325             FROM  
326                 SupportedCaComboConfigs AS t  
327         """
```

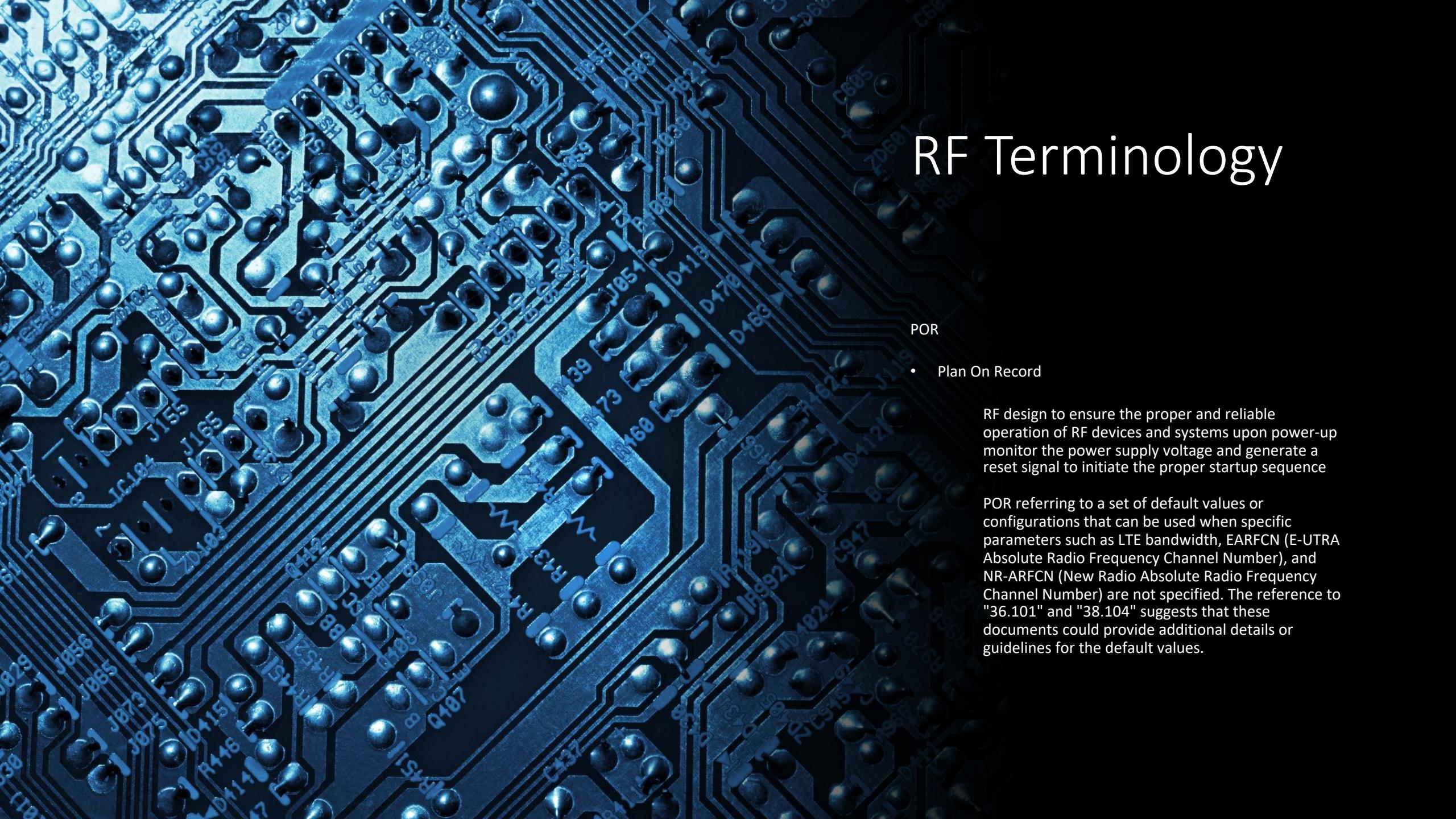
The screenshot shows the Google Cloud Data Marketplace interface. On the left, the 'Schema' tab is selected, displaying the structure of the 'SupportedCaComboConfigs' table. It includes fields like VersionId (Int64), ProjectName (Enum), SkuName (Enum), CarrierName (Enum), ComboKey (String), and ComboConfiguration (Proto). Below the schema, the 'Query Builder' tab is active, showing a list of project names from the table. The 'Query' section contains the code for the class definition:

```
1 SET RequestOptions.requested_enable_feature=LOAD_MISSING_SPANNER_TYPES_F  
ROM_GLOBAL_PROTO_DB;  
2 SET RequestOptions.requested_disable_feature=ENABLE_MAX_TOPOLOGY_DISTANCE_FILTER; # For possible f1::DATA_LOCATION_ERROR:  
http://go/fierror/max_topology_distance_data_location_error.md  
3 SELECT DISTINCT  
4     t.ProjectName  
5 FROM `spanner.global.pixel-modem-tools-modem-config-spanner.prod`.SupportedCaComboConfigs AS t  
6 LIMIT 10;
```



RF Terminology

- L1 RF Emulation
- Responsible for the physical transmission and reception of wireless signals, including the modulation, demodulation, encoding, decoding, and channel coding processes.



RF Terminology

POR

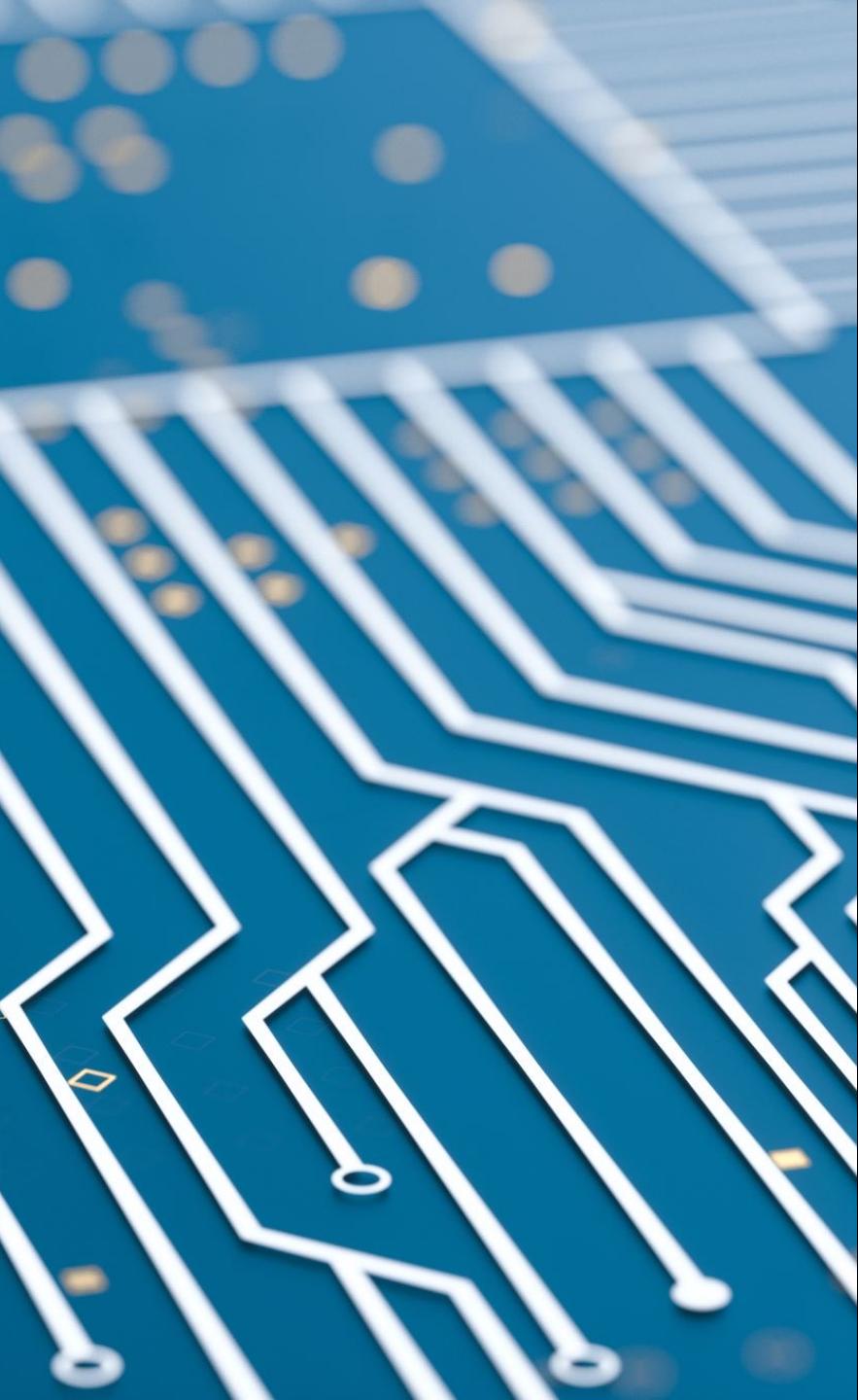
- Plan On Record

RF design to ensure the proper and reliable operation of RF devices and systems upon power-up monitor the power supply voltage and generate a reset signal to initiate the proper startup sequence

POR referring to a set of default values or configurations that can be used when specific parameters such as LTE bandwidth, EARFCN (E-UTRA Absolute Radio Frequency Channel Number), and NR-ARFCN (New Radio Absolute Radio Frequency Channel Number) are not specified. The reference to "36.101" and "38.104" suggests that these documents could provide additional details or guidelines for the default values.

RF Terminology

- E-ARFCN
 - E-UTRA Absolute Radio Frequency Channel Number
 - a parameter used in Long-Term Evolution (LTE) networks
 - Downlink EARFCN
 - Uplink EARFCN



RF Terminology

- 3GPP
 - 36.101 and 38.104
 - Provide guidance and specifications that help ensure the proper design, deployment, and operation of RF communication systems within these cellular networks.
 - Refer to specific technical specifications or standards used in the context of cellular networks, particularly for (3rd Generation Partnership Project) systems like LTE (Long-Term Evolution) and 5G (fifth generation) networks.
 - 3GPP is a collaboration between telecommunications standards organizations that develops global standards for mobile communications technologies, including GSM (Global System for Mobile Communications), 3G (Third Generation), and 4G (Fourth Generation) LTE (Long-Term Evolution) networks. Each release of the 3GPP specifications represents a significant milestone in the evolution of mobile communication technologies.

RF Terminology

- 36.101
 - Provides details about the User Equipment (UE) radio transmission and reception characteristics for LTE networks. It covers topics such as radio frequency bands, channel bandwidths, power control, modulation schemes, coding schemes, and radio performance requirements.
- 38.104
 - Focuses on the radio transmission and reception aspects of 5G NR (New Radio) networks. It provides detailed information about the physical layer procedures and protocols, including radio resource management, channel coding, modulation, and multiplexing techniques.
 - Covers topics such as carrier frequency ranges, channel bandwidths, multiple access schemes, link adaptation, beamforming, and performance requirements for 5G NR systems. It serves as a key reference for implementing 5G networks and ensuring consistent and reliable RF communication between 5G devices and base stations.

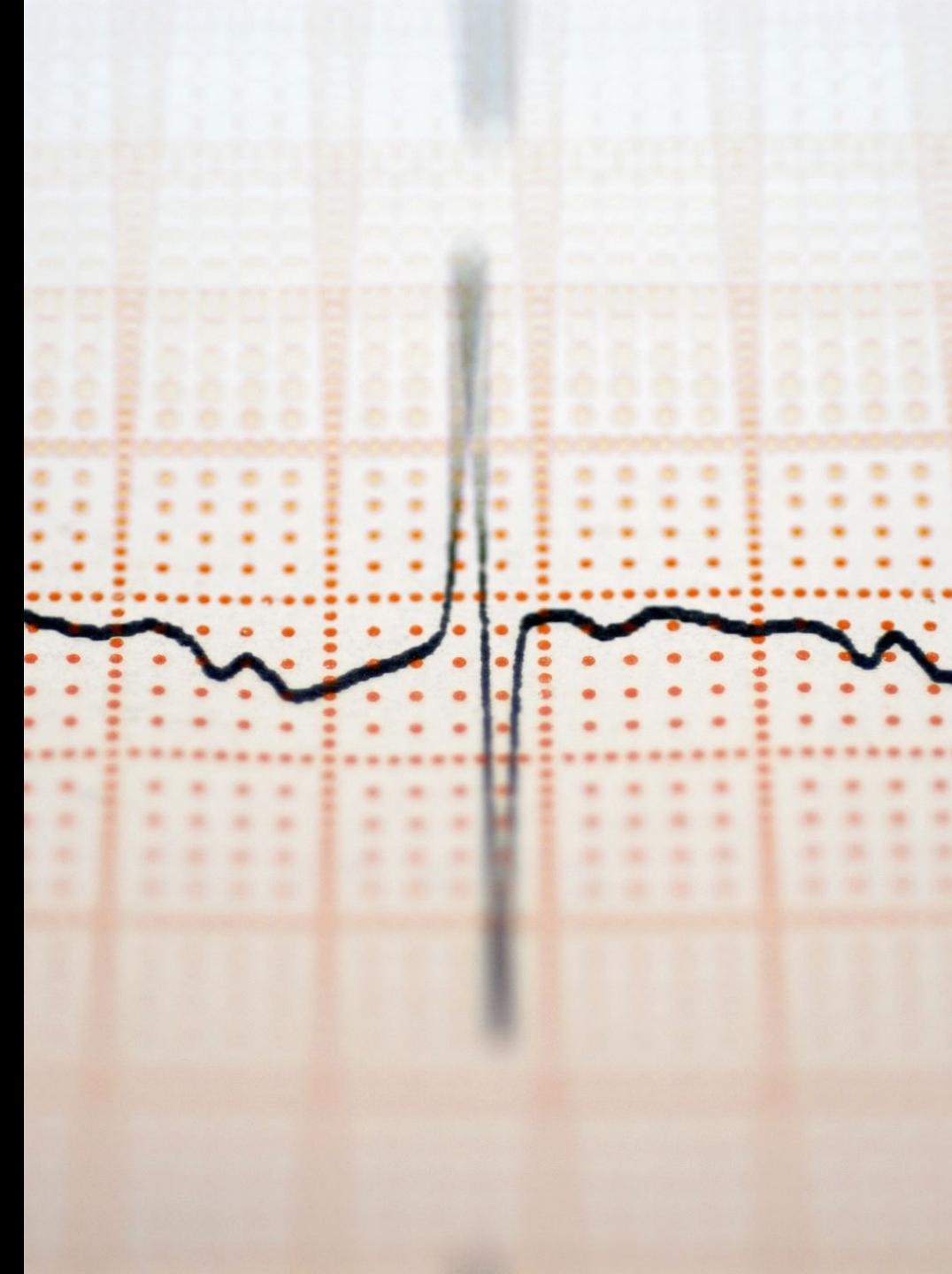
RF Terminology

- L1 RF Emulation
- Responsible for the physical transmission and reception of wireless signals, including the modulation, demodulation, encoding, decoding, and channel coding processes.

RF Terminology

- ARFCN
 - used in 2G networks
- E-ARFCN
 - used in 4G and 5G network
 - The purpose of both parameters is to provide a standardized representation for radio frequency channels, allowing devices to tune into the correct frequency for communication.

It's important to note that the specific frequency associated with an ARFCN or E-ARFCN value depends on the cellular band being used, as different frequency bands are allocated for different regions and network deployments.



RF Terminology

- SKU (Stock Keeping Unit)
- Rest of World (ROW)
- mmW (millimeter wave)
- UL (Uplink)
- MIMO (Multiple-Input Multiple-Output)
- SKU (Stock Keeping Unit)

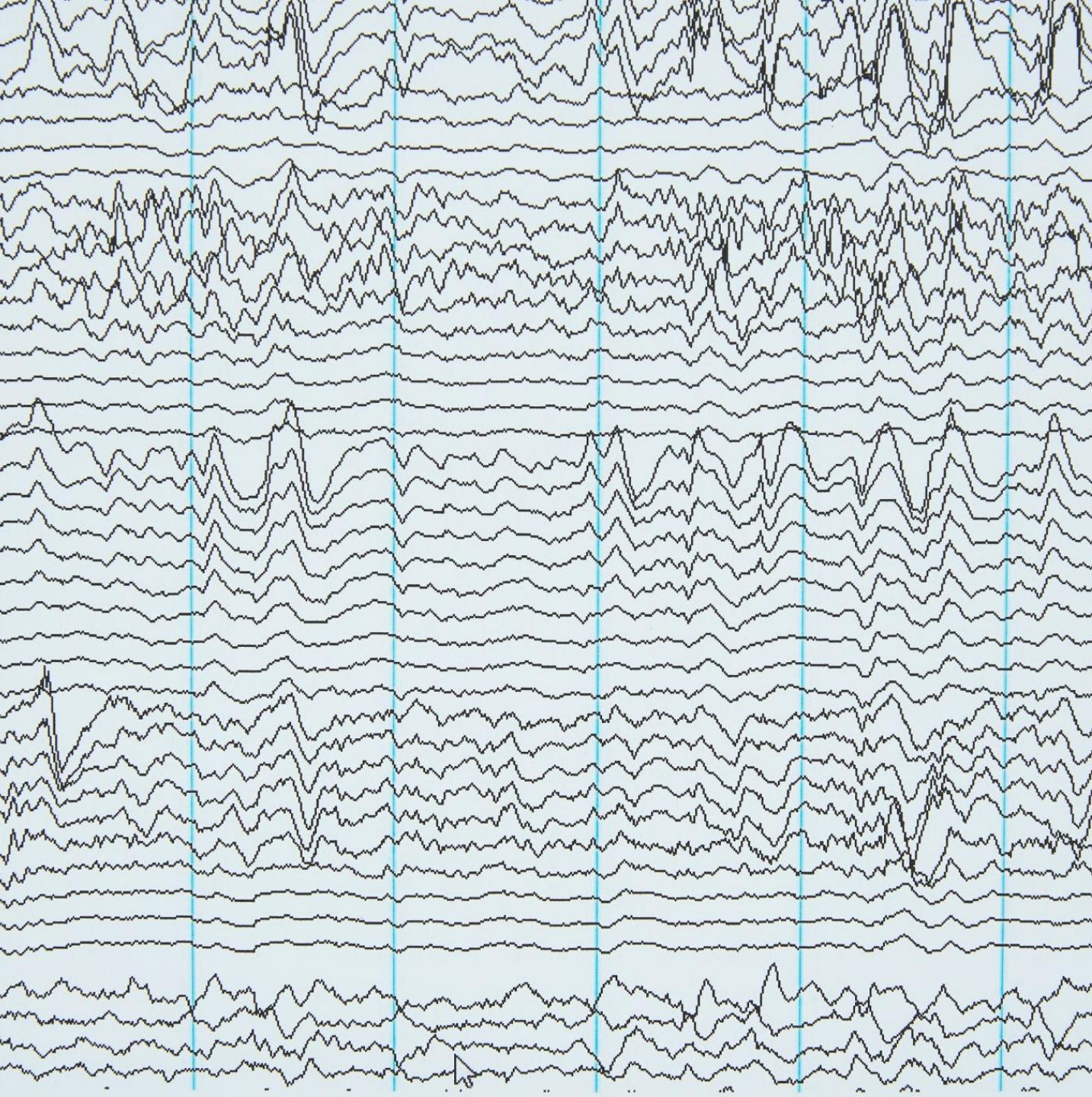


RF Terminology

- US mmW SKU
 - Refers to a Stock Keeping Unit specifically tailored for the United States (US) market that utilizes mmW (millimeter wave) frequencies. Millimeter waves are high-frequency radio waves that offer significant bandwidth for faster data transmission. This SKU is optimized for the mmW frequency range and is typically used for high-speed, short-range wireless communication applications.

RF Terminology

- NA sub6 SKU
 - Stands for North America (NA) sub6 Stock Keeping Unit. "sub6" refers to frequencies below 6 GHz, which includes the traditional cellular frequency bands. This SKU is designed for wireless communication systems operating within the sub6 GHz range, which is commonly used for wider coverage and longer-range communications compared to mmW frequencies. It is typically used in applications such as cellular networks, Wi-Fi, and other wireless technologies.



RF Terminology

- ROW sub6 SKU
 - Represents the Rest of World (ROW) sub6 Stock Keeping Unit. Similar to the NA sub6 SKU, it is optimized for frequencies below 6 GHz and is designed for wireless communication systems deployed in regions outside of North America. It is tailored to the specific requirements and regulations of various countries and regions around the world.



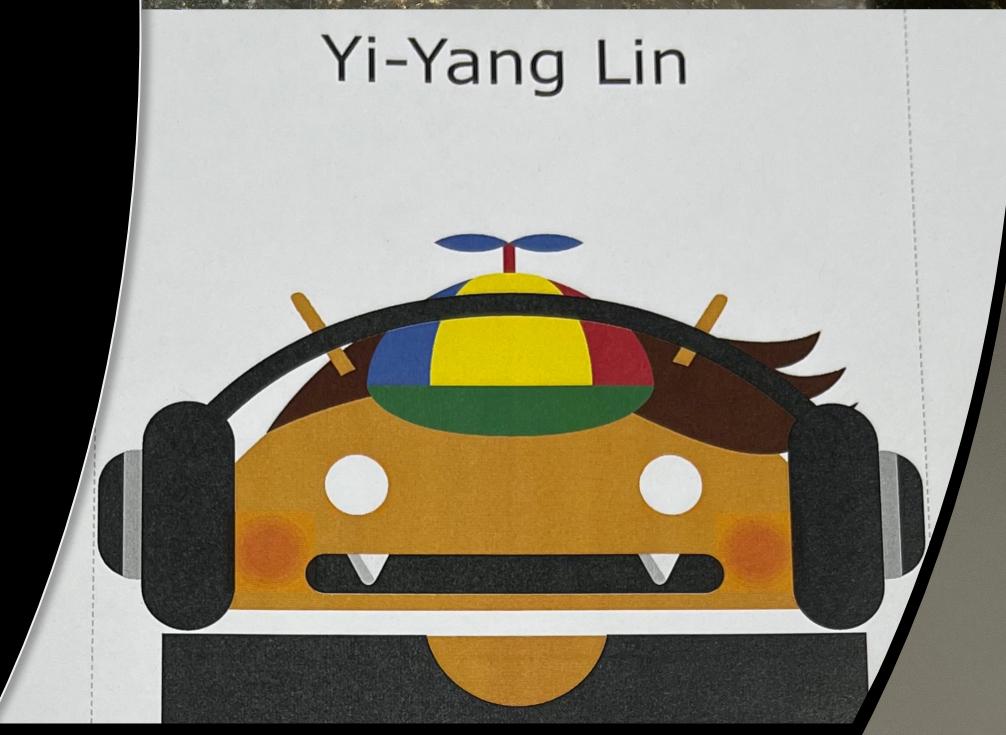
RF Terminology

- SKU (Stock Keeping Unit)
- Rest of World (ROW)
- mmW (millimeter wave)
- UL (Uplink)
- MIMO (Multiple-Input Multiple-Output)
- SKU (Stock Keeping Unit)

Google Life



Onboarding



- focuses on the radio transmission and reception aspects of 5G NR (New Radio) networks. It provides detailed information about the physical layer procedures and protocols, including radio resource management, channel coding, modulation, and multiplexing techniques.
- covers topics such as carrier frequency ranges, channel bandwidths, multiple access schemes, link adaptation, beamforming, and performance requirements for 5G NR systems. It serves as a key reference for implementing 5G networks and ensuring consistent and reliable RF communication between 5G devices and base stations.

ARFCN

- used in 2G networks

E-ARFCN

- used in 4G and 5G network

The purpose of both parameters is to provide a standardized representation for radio frequency channels, allowing devices to tune into the correct frequency for communication.

It's important to note that the specific frequency associated with an ARFCN or E-ARFCN value depends on the cellular band being used, as different frequency bands are allocated for different regions and network deployments.

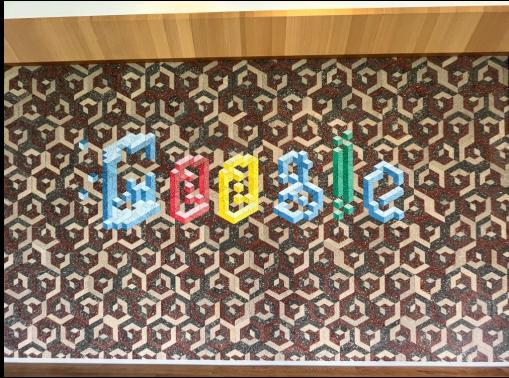
Work with Beer





Study with ChatGPT



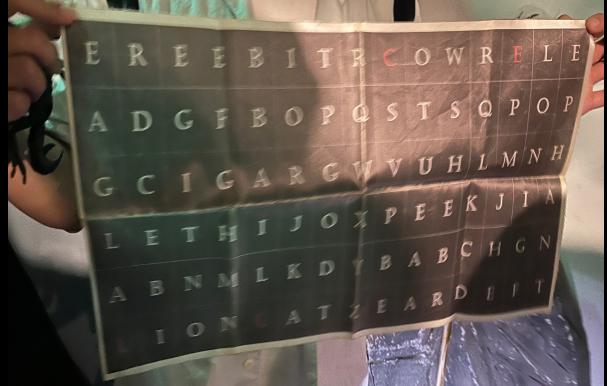






Team Build & Conversation with Managers

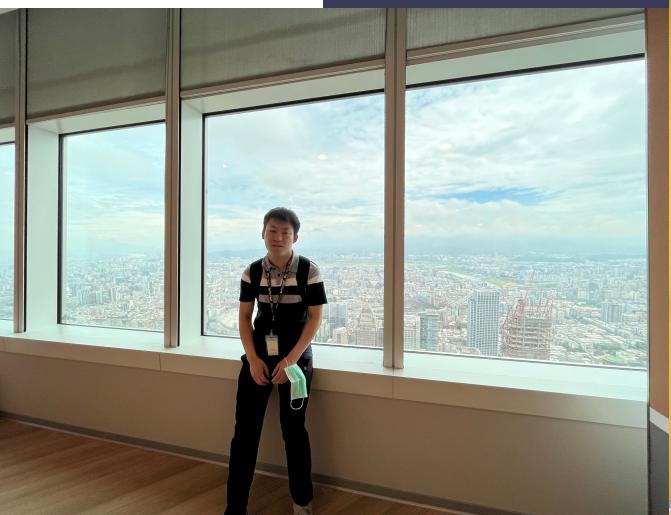


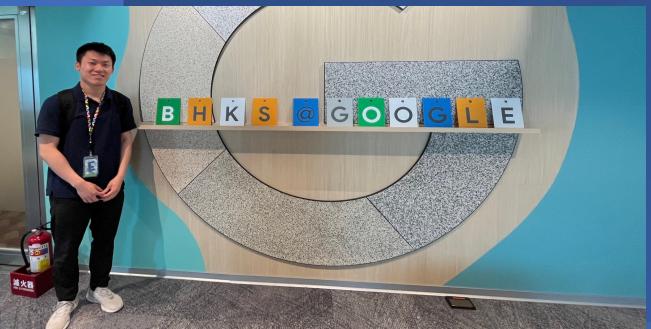
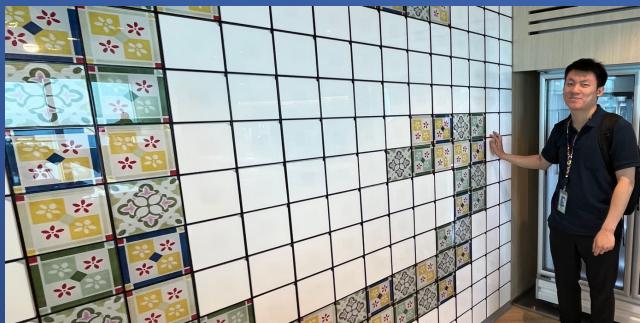


2023 GCN Interns Escape Room Activity



Google Serve – Volunteering Opportunities









TGIF

- Thursday Googler Information Forum
- Company-wide meeting





Thank you

