# C2S2 Analog Subteam
# Analog_FA23_SP24 Repository Guide

Compiled by Daniel Kaminski

## Table of Contents

## Naming Conventions

### Cases and Word Seperation

It will be standard to use an underscore Pascal case naming convention. This means that the first letter of every word will be capitalized, and underscores will be used to signify different words. Some examples are shown below.

This_Is_A_Sample_File_Name.txt (1)

ADC_SigmaDelta_1Bit.mag (2)

Testing_Circuit_SP24.sch (3)

While you do not have to be exact with what counts as a word and what doesn't (for example, "1Bit" or "ADC"), just do what you see to be most sensible. **Sometimes words can be combined, like "SigmaDelta", if they can sensibly go together**. In this case, make sure to capitalize each word. Try to make things as verbose and understandable for others as possible. Please make sure to save your files with this naming convention, and please rename any existing files to match this convention (at least those you're going to upload to the GitHub page).

### File Versioning

To ensure that file versioning is as understandable as possible, please try to follow the following conventions.

File_Name_*Stage*p*Version*p*SubVersion*.Extension (4)

In this case, the "*Stage*" is essentially whether something is in the development stage (0), or if it's ready for production/usage (1), *Version* is how many large changes have been made, and *SubVersion* is simply how many changes have been made since the last major version update. For example, if you're working on a 1-bit $\Delta\Sigma$ ADC that is just starting development in Magic (see .mag extension), you could use the filename in line 5 (line numbers are shown to the right of the content). The "p" is used instead of a "." to avoid issues with certain software.

ADC_SigmaDelta_1Bit_0p0p0.mag (5)

This naming convention indicates that it is a 1-bit $\Delta\Sigma$ ADC that is currently under development, and is in its first revision. When a major change is made, the *Version* number should be increased by 1. It should be noted that these values can go above 9, for example, a valid versioning value is "0p1p23," which means that the project is in development, and there have

been 23 changes since the last major change (another major change would reset the version to "0p2p0").

Relatively speaking, a minor change would be changing how something is routed or re-exporting from magic to LTSpice, while a major change would be adding a new block to a design, or fixing a fundamental flaw. Minor changes would be work that could be completed in about a week, while major changes should occur around every 2-4 weeks. **In general, make sure to save often.**

## Working Tree

### Main Branch

The main branch will always contain the main files for the project. You can switch to it using command 6, where *git* calls the git package, *checkout* "checks-out" to a different source, *-b* specifies that this source is a branch, and *main* specifies that the branch being switched to is the main branch.

```
git checkout -b main
```
(6)

You should more or less never use the main branch unless told to do so by the team lead. The main branch is meant to be an up-to-date presentation of the project, so merging the dev branch with the main branch can introduce a bug from the development branch that hasn't yet been fixed. Anyone who is relying on functional files will see that the main branch has been changed, and on attempting to use the new files will experience issues. So, almost all the work will be performed in the dev branch, which is the subject of the next section.

### Dev Branch

The dev branch is where the working files will live. This is where you should upload all the files you are working with, regardless of whether you think they are important or not. Of course, if something is too large to be uploaded, check with the subteam to see if it should be added. Overall, this branch should be where most of the work goes on, with changes to the main branch only occurring after a major modification is begun, worked on, and polished. The importance of having this backup branch cannot be overstated. To switch to this branch, run command 6, but replace "main" with "dev". Git should then tell you that you have switched over, and running "ls" if you're on Linux or examining your files on other operating systems should show that you are up-to-date with the dev branch.

### FA22_SP23 Branch

The purpose of this branch should be relatively self-explanatory. This is where the testing goes on for the op-amp taped out last year, and all the relevant files, including data and graphs (as well as the python files used to generate those graphs) should be placed here. This is a critical portion of testing documentation, and will be referenced on the testing Cadence page.

## Basic Command Line Interface (CLI) Git Tutorial

A great git tutorial is already on Confluence, so please check that out if you're still confused after this basic tutorial. The linked tutorial also goes over how to set up Git, and how to download a repository, configure your password and username, etc. Of course, there is also a plethora of resources available online, as the git CLI is a very widely used tool. Here, the commands relevant to just this repository and its current branches will be mentioned. The commands from now on will be performed using the Git CLI.

The first command (command 7) is used to clone the GitHub repository (Analog_FA23_SP24) to your current directory. In a bash shell, you can check what that directory is with the command "pwd."

```
git clone https://github.com/cornell-c2s2/Analog_FA23_SP24.git
```
(7)

You can then switch to either the dev (equation 8) or the FA22_SP23 (equation 9) branches, depending on whether you're developing the ADC or testing the OpAmp (respectively). Check that you're on this branch by looking at the files, and seeing that they're matching with what you see on the GitHub page (which can be found at the link in command 7).

```
git checkout -b dev
```
(8)

```
git checkout -b FA22_SP23
```
(9)

At this point, you can perform the work you need to perform using the local files. Whatever you're doing, you first want to run command 10 to check that you're up-to-date, replacing the branch name with whatever branch you're working on.

```
git pull origin BRANCHNAME
```
(10)

Once you're done (and in fact at intermediate points in the git local $\longrightarrow$ remote process), you can run command 11 to check which files have been modified.

```
git status
```
(11)

You can then "add"/stage individual files (equation 12), directories (equation 13), or all of the modified content (equation 14). This tells Git that you'd like to record the changed made to these files. You then "commit" these changes locally using command 15, making sure to add a message stating what you did between the parentheses after the -m flag.

```
git add . FILE1 FILE2 FILE3
```
(12)

```
git add .DIRECTORYNAME/*
```
(13)

```
git add .
```
(14)

```
git commit -m "Type your message here explaning what you did"
```
(15)

Committing is essentially like saying that this is the point at which you'd like Git to take your files, and to store the changes made. This is only done locally, however, so you would then run equation 16, once again replacing BRANCHNAME with the branch you're working on. This command simply sends the files as they were when you committed them to the remote repository stored on GitHub servers, so that others can access your changed files.

```
git push origin BRANCHNAME
```
(16)

You may be prompted for a username and password. If you're using a CLI, there's a chance you may need to generate a personal access token for GitHub, the information about which can be found here. These personal access tokens would be entered in place of your password, and you want to make sure that you keep these somewhere secure, accessible, and permanent. You will likely only need to do this if you have 2FA enabled.

This is simply an example workflow, and you can refer to online tutorials, as well as the tutorial on the C2S2 Confluence page for more information.