Ryan Hansen
Last Edited 5/6/20

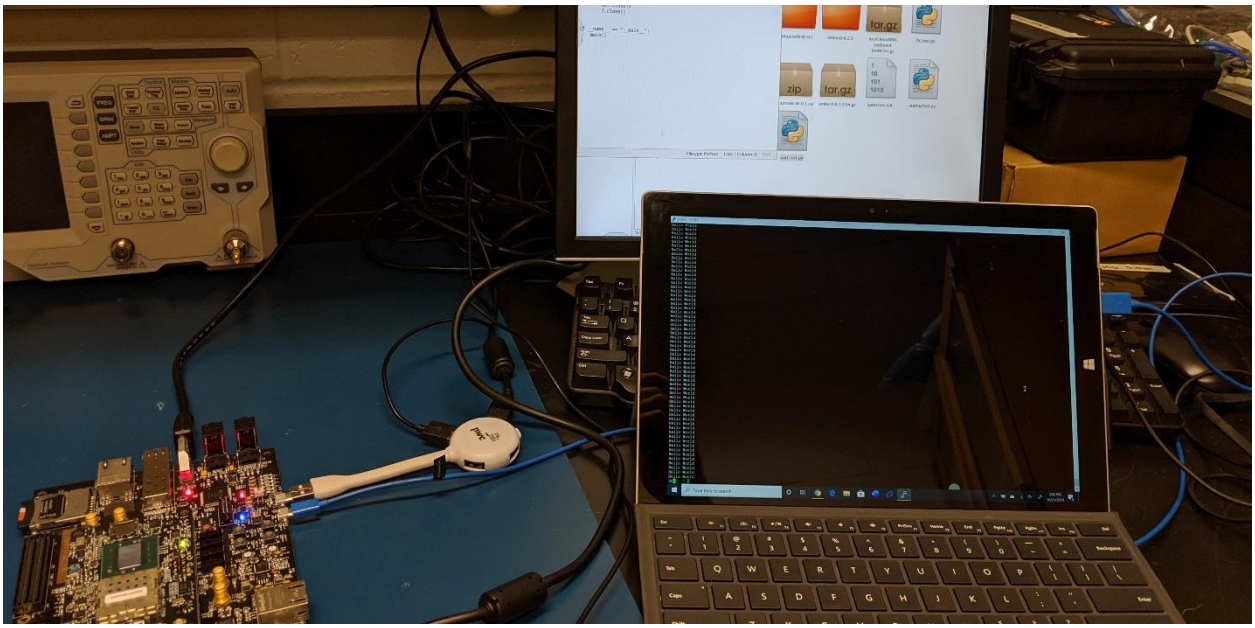
SDR – Overview

Foreword

Ill start this document out by saying if you have not read through the “SDR – Introduction” document stop and read that first as it contains precursor information that might help you understand more about this subsystem.

Setup

Setting up the SDR for work is simple due to its provided development board. First make sure that you have an anti-static mat and wrist strap available. Do not attempt to setup the SDR without those items or bad things/events might occur. The SDR is to be stored on the development board so that the SDR connectors do not get worn out. Locate the white box that says FPGA or similar on it, it should be about the size of a small shoebox. Take out the SDR development board and already attached SDR card. Get out the power cord with power brick and plug into the wall. Make sure that the power switch on the SDR is in the off position and plug the power cord into the development board. Next you will need a computer monitor with HDMI input and an HDMI cord. Plug the HDMI into the monitor and the SDR development board. You will need a USB hub to connect and mouse and keyboard that has an adapter to plug into micro USB. There are two micro USB ports on the SDR development board. One is OTG or USB On The Go. Going to be honest not entirely sure what that means but that’s the port you want to connect the mouse and keyboard into. The other port is for serial communication or more specifically UART. Plug a laptop into the UART micro USB port and open puTTY. Figure out the port number and the baud rate is 115200. Finally, turn on the power switch on the SDR development board and you should see some red and green lights turn on. Shortly after turning on the board a blue light should become lit up. This means that the FPGA has programed the configurable logic. Linux should soon start booting up immediately on the computer monitor. Setup should look like the picture below.



Ryan Hansen
Last Edited 5/6/20

Linux

I am by no means a Linux person and honestly struggle using it sometimes. When I was handed this subsystem there were a lot of things in the Linux system that where still set up that were causing issues. I had to spend time figuring those out and setting things up so that the core of this system would work well with integration into the whole satellite. That said, the folder structure is a bi of a mess since multiple groups have worked on this subsystem now not to mention a few more singular individuals. I would encourage you to explore around the folder structure a bit as there are some interesting things in there. The main folder that you should be working out of is in the home directory that is called “analog”. The folder where my work was placed was within the sdr_dev folder.

Disclaimer, unfortunately I do not have a great understanding of the folder structure anymore due to not being able to work on the physical hardware since February of 2020 due to COVID-19. Even before that I had stepped over to other subsystems to assist and therefore have not done any more development on the SDR since the end of January or early February. So, I will not be able to give you a good direction to exact folder or file contents when I am talking about them from here on out.

Communication

The SDR is a subsystem of the entire CySat system. For the SDR to operate it needs to communicate with the OBC or Onboard Computer. The SDR component is a little odd as it is one of the only components to utilize only UART for its communication to the OBC. The SDR will act mostly like an I2C slave only sending data when asked otherwise it remains silent and just handles commands. As mentioned, the commands will come from the OBC formatted in the CySat Packet Protocol. Some of the necessary commands have been laid out and others will still need to be set up. If you are unfamiliar with the CySat Packet Protocol you should read up on that documentation as it will be important going forward. The eventual plan for the SDR was to also use its own I2C network and act as a master to talk with components on the LNA board which would be I2C slaves. This was not competed or test during my tenure on the project. For the most part, however, the basis of the communication infrastructure is complete, and the only control flow and command handling needs to have continued development.

Control Flow

You will hear me talk about control flow throughout the documentation and whether that’s the right terminology I’m not sure, but it seems to sound right to me. I did lot of work getting the control flow to a point that it can do a lot of the functions it needs to do. I’d say I got it more than half done of it done at the time I worked on it up until around February and then after that we started throwing around some new ideas it fell to being a little under half done in my opinion. Listed below is an overview of the communication and what should happen between the SDR and OBC. There are two operation modes for the SDR and so I have separated the flows accordingly.

Capture Mode: Used to capture data from the radiometer and process it into digital files

1. SDR is turned on via the EPS and OBC
2. OBC asks for power status from the SDR
3. OBC sends packet with time information to SDR
4. SDR time is updated

5. OBC sends packet with parameters for capture script
6. SDR starts script recording data for parameterized amount of time
7. SDR saves data file once complete
8. OBC asks SDR to shutdown
9. OBC turns off SDR

Transfer Mode: Used to transfer captured digital files to OBC and then transmitted to ground station.

1. SDR is turned on via the EPS and OBC
2. OBC asks for power status from the SDR
3. OBC asks SDR to transfer files
4. SDR transmits files line by line to the OBC and that is relayed via the UHF antenna to Earth
5. SDR sends file end command
6. Ground Station confirms it received file
7. Steps 4-6 loops for all files on SDR
8. SDR deletes the files that are confirmed delivered
9. OBC asks SDR to shutdown
10. OBC turns off SDR

This is a very simplified list of the commands sent in the different modes. More detailed information is found in the document folder for the SDR Board specifically in the "GS_OBC_SDR Communication" and "OBC_SDR Communication" files.

GNU Radio

So the main software defined radio part of this project has already been completed. It was done by Matthew Nelson for this thesis paper. He used a program called GNU Radio Companion and this program allows you to build processes with blocks and connect them together to run tasks. It's like cookie cutter online coding projects that you may have done in high school. Anyway, this program allows you to export your defined operations to a python script. This has already been done and I have used it and modified it slightly to change the location of the files that it saves. The file "AD9361_TPR_nogui.py" is the file that has been generated from the GNU Radio application. Feel free to open that file with GNU Radio and look around at what it is doing from an overview perspective. The code I have developed uses that file and plus in the parameters to make it run.

Development

Due to COVID-19 I was not able to get the most recent working files upload to GIT so I am going off memory here. There are two files that are the most important for development. The first being the file that will run on the SDR while it is in space is called "startup_radiometer_app.py" and it contains a lot of my development work. The second file is called "startup_radiometer_app_driver.py" and this is a file that I used for testing. The "startup_radiometer_app_driver.py" file is to be ran on a laptop connected to the SDR via the UART micro USB. It will simulate the commands that the OBC will send to the SDR and is useful for testing. Its not the best but it does its job for initial testing. Within these files I have created a host of functions and commented them with what they do. There is another file called "packet_utilities.py" which also has these functions and can be thought of as a header file for C. It serves no purpose other than holding those functions for documentation.

Again, GIT does not have the most recent version of the “startup_radiometer_app.py” file because that development was done of the physical board which does not have internet access without using an ethernet cable plugged into the only ethernet plug in the M2I lab that I know of that works (it’s across the lab by the solder stations). So, uploading changes to GIT can be a bit of a hassle if you don’t have a flash drive on you.

That all said, from here on out I am going by my last memory of working with the SDR. You should be able to start the python script “startup_radiometer_app.py” in a terminal on the SDR and wait for it to print some initial information and then after that start the “startup_radiometer_app_driver.py” script on your laptop in a terminal and see things happening on both of the devices. The SDR should respond since its power is always on (there wasn’t a good way to simulate this without setting the script to run at startup and physically turn the development board on and off). Then the time should get set on the SDR to whatever the time is on your laptop. After that, the capture script (AD9361_TPR_nogui) should begin running with the parameters that were sent to it. This will continue for a bit and then the capture script should stop and return to the “startup_radiometer_app.py” script. There may be a line commented out about shutting down the SDR if not that will need to be added. I’m pretty sure there is an example of that in the code somewhere.

There should be a folder named “data” or something like that in the same directory where the captured data files should be placed. They are configured to save with the date and time and are in a .dat format. This format cannot just be opened in Linux because the files have not text headers in them. It’s literally just 32bit floating point numbers on lines so it has no information on how to open them. Look around on the internet for how to open these files as I could really find the best method in my opinion. One way is to use Scipy which is a python library to read line by line. If I’m remembering correctly there might even be a file named “data_open” or something like that with an example, remember this is all pre COVID-19 work.

So if I am remembering correctly the capturing mode should be almost complete or completed. The transfer mode needs a majority of the work and ultimately, I stopped working on that because we had started discussing adding a sequence number to our CySat Packet Protocol and that would have affected the operation of that mode. Probably making it more robust but would have needed a lot better control flow. So, with this information you should be able to get things working or set up.

Handoff

I have set up a “SDR – Next Steps” document to summarize a few things and list some stretch goals. Older information that might have some helpfulness will be under the folder path “/CubeSAT/other/CprE 488 Team Work” in the GIT repository. It has information from the previous team that worked on the SDR and OBC before my team so keep in mind it might be outdated in some sections. My name is Ryan Hansen and you can find my contact information in the GIT repository if you need to reach out about any specifics.