Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

Date: 5/18/2021

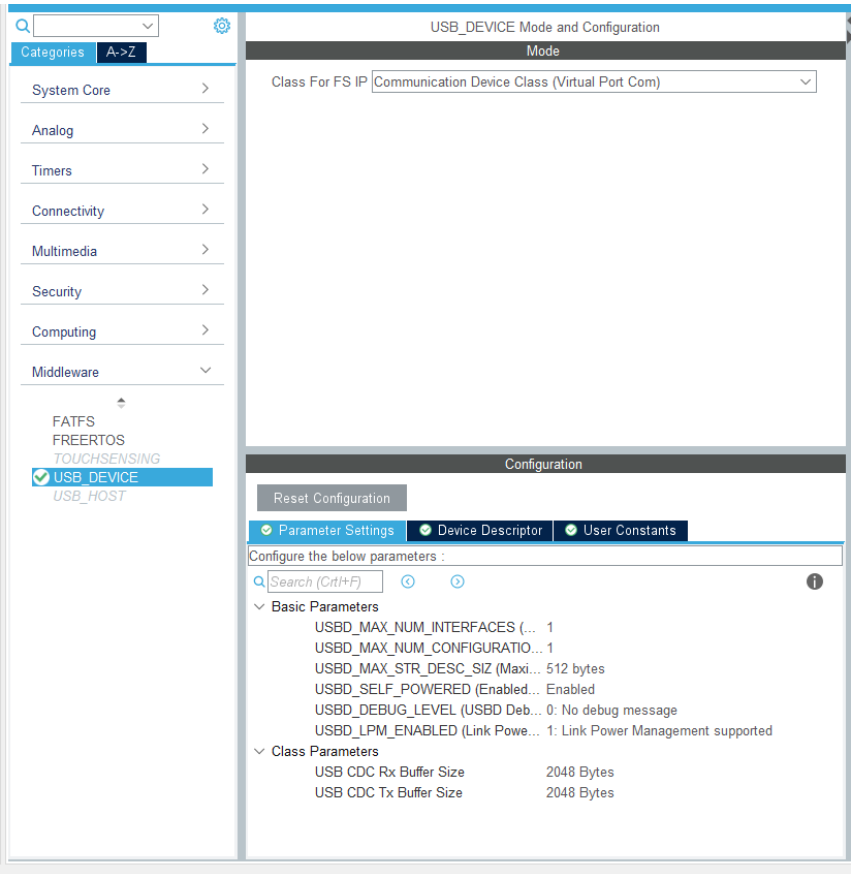
Assignment 6: USB Hands On

The following will document completion of the sixth assignment for ECE-40293, using the onboard hardware to complete the following user stories:

1. **USB Middleware. Use STM32CubeIDE to add USB Middleware to create a Virtual Com Port.**
2. **Plug into host and see COM port appear. When you plug the USB device into a USB Host you will see a new COM port appear!**

1. USB Middleware. Use STM32CubeIDE to add USB Middleware to create a Virtual Com Port.

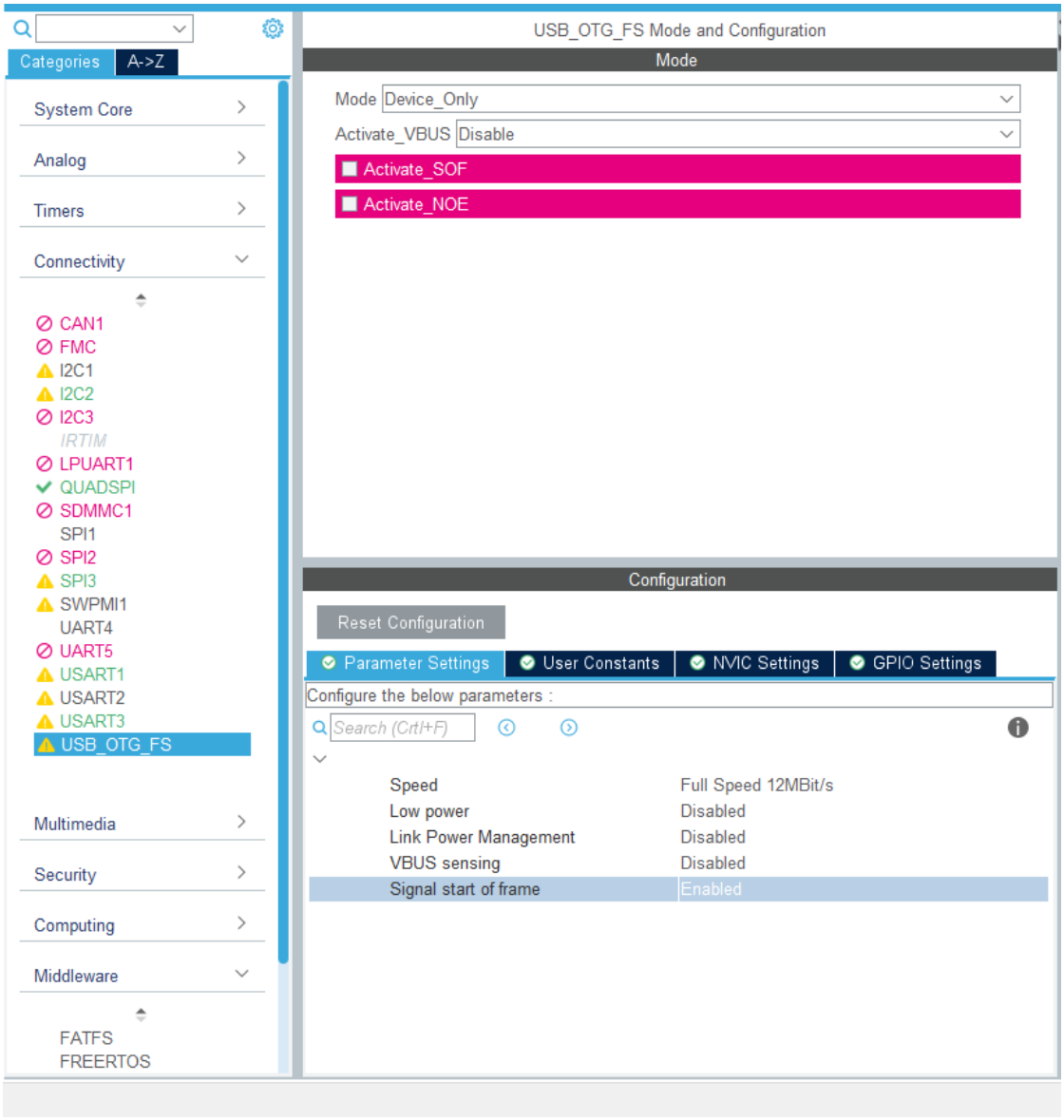
To begin, we will generate a new project as in other assignments using the default configuration methods. Once the project is built, we will open the configurator and browse to the peripheral config interface and expand the middleware selection, choosing “USB_DEVICE”. In the associated window, we will want to select the option for a virtual comm port form the drop-down menu, leaving the default selections in place.



Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

Date: 5/18/2021

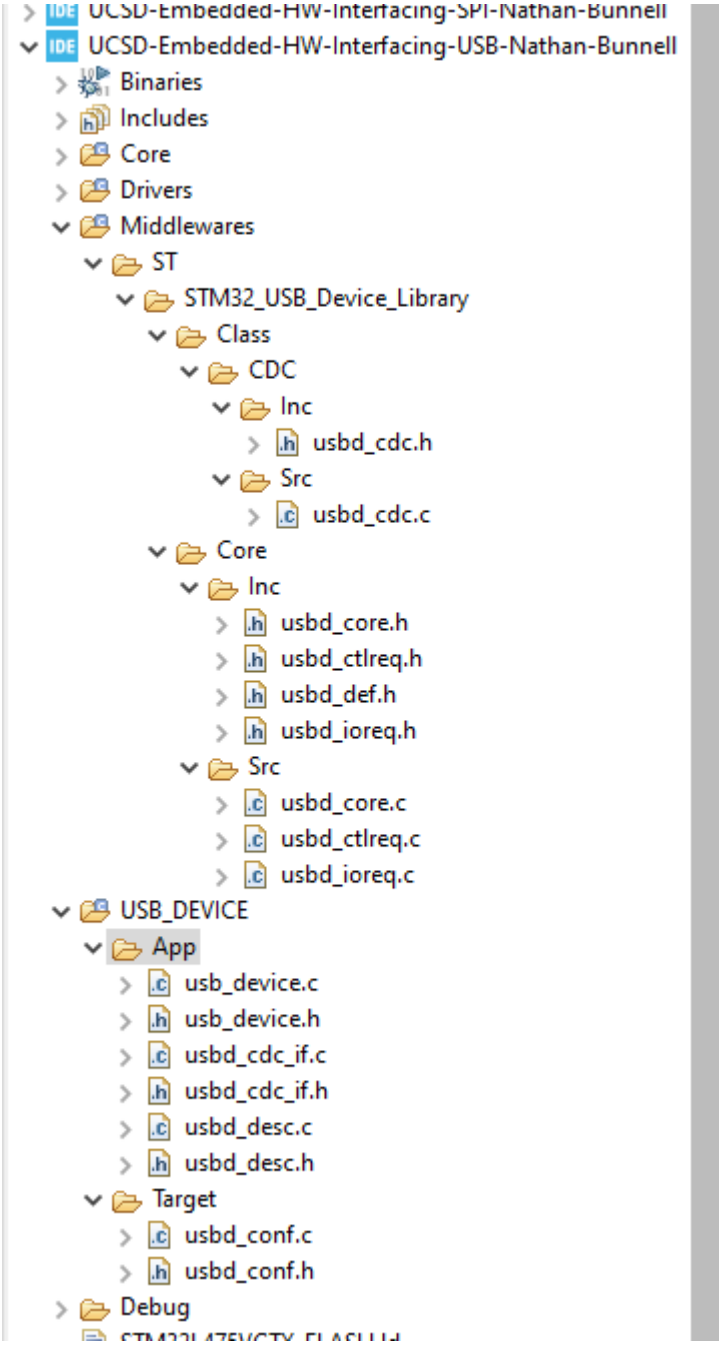
As a next step, we will need to browse to the Connectivity peripheral area and select “USB_OTG_FS”. Within that configuration window, we will need to ensure that the option for “Signal start of frame” is set to enabled.



Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

Date: 5/18/2021

At this point, we can close the project configuration and allow it to rebuild with the necessary middleware and USB files being added automatically. Once it is complete, we should see something like the following in our project structure:

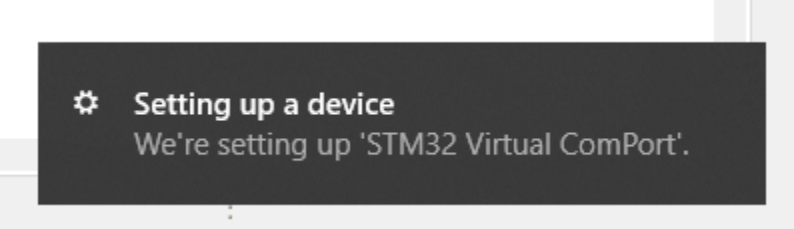


Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

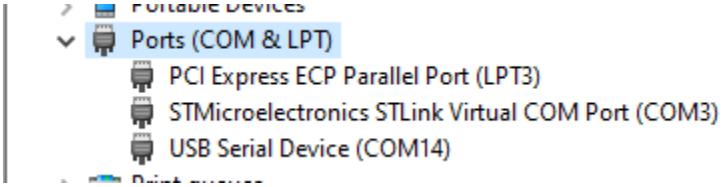
Date: 5/18/2021

2. Plug into host and see COM port appear. When you plug the USB device into a USB Host you will see a new COM port appear!

At this point, we can compile and flash the program to our disco board and connect to the OTG port. At the time of testing this, I forgot about the power selection jumpers on the board and was at first puzzled to see that the OTG wasn't supplying power. I left both USB ports connected to my hub to work around this and realized the source of the issue while drafting this report the following morning. At any rate, with the OTG port connected to a Windows 10 PC, I saw the following notification appear, indicating success:



And, under the Device Manager, we see the ST-Link and the COM port for the OTG connection:



Then for good measure, I also connected to one of my Linux boxes and tested with lsusb, to find similar results:

```
nathan@briarBox:~/UC_SD_ESE/Spring2021/EmbeddedSystemsHardwareInterfacing$ lsusb
Bus 002 Device 005: ID 0bda:0411 Realtek Semiconductor Corp.
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 004: ID 04ca:3007 Lite-On Technology Corp.
Bus 001 Device 015: ID 0483:5740 STMicroelectronics STM32F407
Bus 001 Device 014: ID 0483:374b STMicroelectronics ST-LINK/V2.1 (Nucleo-F103RB)
Bus 001 Device 013: ID 0bda:5411 Realtek Semiconductor Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Closing thoughts

I have to say, I was really surprised by how easy this was to accomplish. I think from start to finish, this process took about 10 minutes, with a significant portion of that time being related to the IDE loading and generating code. Granted, this doesn't really do anything useful, but it seems that a lot of the heavy lifting is taken care of for you upfront as a developer. My other takeaway from this exercise is that I would really like to dive deeper into what is happening beyond the middleware wrappers. I recall

Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

Date: 5/18/2021

an Adafruit tutorial from several years ago that specifically said “USB is hard” and at the time, it scared me off as a newbie. Later, when the Raspberry Pi Zero was released, I found another article about how the device could be configured as a serial or ethernet over USB as a “gadget” and caught a glimpse of the whole world of complexity in the Linux kernel that drives that capacity. I still only have an extremely minimal understanding of the protocols behind this function and hope to change that in the future.