AeroSimMCU

C code for STM32 Microcontroller interfacing with FPGA (via SPI) and a host computer with the AeroSim user interface (via USB)

Projects in this Repository

There are several pieces of firmware and supporting code in this repository:

- **AeroSimFW:** The primary firmware for the microcontroller of the AeroSim interface device, configured for the STM32G431KB microcontroller used on the custom interface device PCB

> **NOTE:** The AeroSimFW project is set up for revA of the AeroSim interface device hardware. To use the revB hardware, the configured pinout will require modification.

- **prototypeSPI:** A development project for testing the FPGA breakout board by transmitting TX commands on SPI and printing RX data to a serial terminal via UART; this project is configured for an STM32F401RE Nucleo-64 board that is wired up to the SPI pins of the FPGA board as a stand-in for the actual interface device PCB (that has the microcontoller running AeroSimFW)

- **fakeFPGArx:** Another development project, this one for testing the RX datapath functionality of the actual interface device microcontroller (running AeroSimFW); this project is configured for an STM32F401RE Nucleo-64 board that is wired up to the interface device FPGA socket SPI and interrupt pins to stand in for the FPGA board

- **AeroSimSerial** (in `tools/`): Basic python script for generating and sending commands to the interface device from a host computer over USB Serial communication during development; note that the command formatting is common to both USB serial and SPI communication in the system (except for discrete output commands which are handled by the MCU, not transferred on the SPI bus to the FPGA). Edit the `SERIAL_PORT` string to match the serial/COM port for the interface device on your system (eg. `SERIAL_PORT = "COM3"`)

AeroSim Interface Device LED Behaviour

There are two LEDs on the interface device. The first is a green LED that indicates if the device has power. The second is a red LED that is controlled by the microcontroller to indicate its status.

The red status LED may signal states according to the following list. Note that the state of the interface device's FPGA does not have any influence on the behaviour of this LED. A fault or breakdown of the FPGA's operation (due to any uncaught bugs in the FPGA logic) will not be signalled by this LED.

- **Off:** The interface device is initializing. It should enter normal operating state (LED blinking quickly) after about 6 seconds (this is the time it takes for the FPGA to load its programming configuration from flash memory).

- **Blinking quickly (5Hz):** The microcontroller is operating normally.

- **Blinking slowing (1Hz):** The microcontroller is faulted. If this happens, it is recommended to unplug the interface device from USB and reconnect it to power cycle the device. When faulted, the interface

device will not accept any commands from the PC software and will not return any data. If ARINC transmission was ongoing when the fault occured, it may continue with the set configuration until the device is unplugged.

> **DEV NOTE:** The most likely fault typically happens in response to a buffer overflow. Eg. the interface device is actively receiving ARINC data from avionics equipment and the PC software improperly terminates, at which point the PC stops the intake of data from the interface device without turning the RX functionality off. RX data piles up in the microcontroller's data buffer, eventually reaching an overflow.

IDE Setup

This project uses STM32CubeIDE for embedded programming.

Download: STM32CubeIDE. Turn off any adblockers and enable cookies for the download to work. Try a different browser if it doesn't work. Use 10minuteMail when it asks for an email if you like. Note that you have to follow a download link that is emailed to the email address entered to get the download.

To open the project after cloning the repository, in the menu bar click `File` > `Open Projects From File System...` and click `Directory...` to point the IDE to the location of the project folder in the repository. Then, click `Finish`.

Working with the Firmware

Code Generation

Boilerplate STM32 HAL and USB library code implementing the desired peripheral hardware configuration and clock configuration for the microcontroller is generated using STM32CubeMX (within STM32CubeIDE). To change things like microcontroller pin function assignments (pinout) and USB device identification, this tool must be used to regenerate the boilerplate portions of the project. Open the `AeroSimFW.ioc` file in the IDE to edit this configuration and regenerate code.

Building the Firmware

To compile the firmware project, open any source code file in the project in the IDE. Then, in the menu bar click `Project` > `Build Project`.

Flashing the Firmware

To flash the firmware to the interface device, connect an ST-LINK programing/debug probe to the interface device's SWDIO, SWCLK, and GND pins and optionally connect the 3V3 pin to 3.3V power if the interface device is not plugged into a USB port and getting power from there. You can use either a standalone ST-LINK device or the ST-LINK integrated with an STM32 Nucleo-64 development board; see the STMicroelectronics Nucleo board manual UM2324 section 6.4.4 for more information.

With the ST-LINK connected to the interface device correctly, plug the ST-LINK into a USB port.

In STM32CubeIDE's menu bar, click `Run` > `Run`. The first time you do this, you will be asked to create a launch configuration. The defaults should be fine. Click the `Run` button at the bottom of the popup window. The code will then be compiled and the IDE will attempt to flash the compiled binary to the interface device

via the ST-LINK. The progression of the flashing sequence will display in a terminal window. If any problems are encountered, an error popup is shown. Otherwise, the firmware should have been successfully flashed to the interface device. The red LED on the interface device should start blinking quickly after several seconds.

Flashing the Firmware from Hex File

A hex file firmware binary is included in this repository: `AeroSimFW/Binaries/AeroSimFW-revA.hex`

This binary was built just prior to project handoff. Flashing the firmware this way may be quicker for non-developers.

To flash this firmware binary, connect an ST-LINK programing/debug probe as described above. Then, use STM32CubeProgrammer (which is a piece of software downloaded separately from STM32CubeIDE) to flash the hex file firmware to the interface device.

Make sure "ST-LINK" is selected as the connection method and click the `Connect` button. On the left, click on "Erasing and Programming" (green download icon). Enable `Verify Programming` and `Run after programming`. Select the hex file for the file path. Click `Start Programming`. When done, click `Disconnect`. The red LED on the interface device should start blinking quickly after several seconds.

Editing the Firmware in VS Code

This setup is rather fragile. Use STM32CubeIDE for the most reliable experience.

`c_cpp_properties.json` and `tasks.json` VS Code configuration files are respectively provided to get Intellisense to work with the CubeIDE project and build the project using STM32CubeIDE provided tools. Note that the toolchain paths in these may need modification after updates to STM32CubeIDE. Also note that these configuration files have not been completed for Windows.

With the `task.json` aligned to the STM32CubeIDE toolchain paths, pressing `CTRL`/`CMD` - `SHIFT` - `B` in VS Code will build the project identically to STM32CubeIDE. The first time doing this, and any time the build configuration is modified (eg. files to compile are added or removed), you must build the project once in STM32CubeIDE before initiating a build from VS Code. The VS Code task relies on the makefile and other resources auto-generated by STM32CubeIDE in its build folder.