Needle_controller-mbed

Mbed FRDM K64F with Mikroe ADC 18 Click!

Read multi-channel, wide voltage-range differential data with an mbed microcontroller.

This is a C++ driver for an mbed FRDM K64F device equipped with a Mikroe Arduino Uno Shield and Mikroe ADC 18 Click board, which is designed to handle all the sensor measurements in a bespoke laboratory needle-insertion test system. However it can be adapted for other applications requiring reading multiple analogue input channels with wide voltage-range (+/- 10 V) at fast data rates, and/or reading multiple encoder values.

The driver is designed to be slave to a host device which sends and receives commands via Ethernet TCP/IP. See `Needle_controller_tester.py` in the project root for an example of how to establish a connection and parse commands to the controller.

Description

A parent `NeedleController` class interfaces with a 24-bit Analog-to-Digital Converter (ADC) via the `ADC18` class and three encoders using a Quadrature Encoder Interface (QEI) via the `QEI` class.

In the parent `NeedleController` class, the pins used to construct an instance of the `ADC18` class correspond to the pins for position 1 of the Mikroe Arduino Uno Click Shield. With the exception of the ready pin `rdy` which is ported from the default analogue pin (A0) to a digital pin (D7) by a simple external circuit, for greater determinism during falling edge detection.

The `ADC18` class file is adapted from the official Mikroe source code.

A `QEI` class file is also included to simultaneously read encoder data with QEI documented here.

Hardware Requirements

- Mbed FRDM K64F board
- Mikroe Arduino Uno Click Shield (MIKROE-1581)
- Mikroe ADC 18 Click board (MIKROE-5132)

Installation and Compilation

Clone the repository:

```
bash git clone https://github.com/WillEStokes/Needle_controller-mbed.git
```

This project uses an offline version of the mbed library which is configured for the FRDM K64F target using the instructions in the mbed-cmake repo.

To configure the build system and compile the project:

1. Get CMake by downloading the installer from here. Make sure to select "Add CMake to the system PATH for all users" during installation!
2. Get the GNU ARM toolchain by downloading the 32-bit installer from ARM's website. Make sure to check "Add path to environment variable" during installation!
3. Get the latest release of Ninja by downloading the exe from here.
4. If necessary, update PATH variables so that CMake, GNU ARM toolchain and Ninja are visible on your system.
5. To configure cmake build files for the project, run `cmake -G Ninja ${Workspace Folder}` in the command console from the build directory. Make sure to terminate ${Workspace Folder} with a "\". Previous cache files may need to be manually removed before configuring, or reconfiguring any build files.
6. Finally to compile the project into a binary file, simply run `ninja` in the command console from the build directory.

API Reference

1. NeedleController Class

The FRDM K64F device flashed with this driver interfaces with a 24-bit Analog-to-Digital Converter (ADC) and three encoders using a Quadrature Encoder Interface (QEI). The driver provides methods for getting system status, sensor data, starting and stopping data acquisition, and configuring the ADC.

Usage

To use this driver, you need to create an instance of the `NeedleController` class, passing the pin names for the red LED and status LED to the

constructor.

```
cpp NeedleController needleController(PinName redLED, PinName statusLED);
```

Then, you can use the methods provided by the `NeedleController` class to interact with the device.

```
cpp needleController.run();
```

Control Interface

This class contains a message array variable `comMessages`, which holds pointers to several private methods, each assigned a unique function ID.

When connected to a host device, the run loop continuously checks for incoming `MessageHeader` messages containing the ID of the method to call. Upon receiving a message, the loop identifies and invokes the corresponding method based on the function ID in the `comMessages` array.

```
cpp typedef struct { uint16_t packetLength; uint8_t fid; uint8_t error; } __attribute__((__packed__))
MessageHeader;
```

- `packetLength`: Length of the message packet in bytes.
- `fid`: Unique function ID.
- `error`: Error status.

Private Methods

This class contains private methods for getting system status, system info, force-torque sensor data, encoder sensor data, all sensor data, starting and stopping data acquisition, resetting the ADC, checking the ADC, setting the ADC conversion mode, and setting the ADC data rate.

- `void getStatus(const MessageHeader* data)`: Retrieves the status of the device.
- `void getSystemInfo(const MessageHeader* data)`: Retrieves the system information.
- `void getFTSensorData(const MessageHeader* data)`: Retrieves the Force-Torque sensor data.
- `void getEncoderSensorData(const MessageHeader* data)`: Retrieves the encoder sensor data.
- `void getAllSensorData(const MessageHeader* data)`: Retrieves all sensor data.
- `void getAllSensorDataMean(const Settings* data)`: Retrieves all sensor data multiple times based on the settings.
- `void startAcquisitionStream(const MessageHeader* data)`: Starts the acquisition stream.
- `void stopAcquisitionStream(const MessageHeader* data)`: Stops the acquisition stream.
- `void resetADC(const MessageHeader* data)`: Resets the ADC (Analog-to-Digital Converter).
- `void checkADC(const MessageHeader* data)`: Checks the status of the ADC.
- `void setADCConversionMode(const Settings* data)`: Sets the conversion mode of the ADC.
- `void setADCDataRate(const Settings* data)`: Sets the data rate of the ADC.

Function IDs

- `FID_GET_STATUS = 0`: Used to get the status of the device.
- `FID_GET_SYSTEM_INFO = 1`: Used to get the system information.
- `FID_GET_FT_SENSOR_DATA = 2`: Used to get the Force-Torque sensor data.
- `FID_GET_ENCODER_SENSOR_DATA = 3`: Used to get the encoder sensor data.
- `FID_GET_ALL_SENSOR_DATA = 4`: Used to get all sensor data.
- `FID_GET_ALL_SENSOR_DATA_MEAN = 5`: Used to get the mean of multiple sensor data readings.
- `FID_START_ACQUISITION_STREAM = 6`: Used to start the acquisition stream.
- `FID_STOP_ACQUISITION_STREAM = 7`: Used to stop the acquisition stream.
- `FID_RESET_ADC = 8`: Used to reset the ADC (Analog-to-Digital Converter).
- `FID_CHECK_ADC = 9`: Used to check the status of the ADC.
- `FID_SET_ADC_CONVERSION_MODE = 10`: Used to set the conversion mode of the ADC.
- `FID_SET_ADC_DATA_RATE = 11`: Used to set the data rate of the ADC.

Requirements

This driver requires the mbed.h, EthernetInterface.h, ADC18.h, and QEI.h libraries.

2. ADC18 Driver Class

This is a C++ driver for a Mikroe Analog-to-Digital Converter (ADC), specifically the 24-bit ADC 18 model. This driver allows you to interact with the ADC, providing methods for setting the conversion mode, setting the data rate and reading the ADC data.

Features

- Set conversion mode

- Set data rate
- Read ADC data (single data point)
- Read ADC data (mean of user-specified number of data points)

Usage

To use this driver, you need to create an instance of the `ADC18` class, passing the pin names for the ready signal, chip select, interrupt pin, MOSI (Master Out Slave In), MISO (Master In Slave Out), and SCK (Serial Clock) to the constructor.

```
cpp ADC18 adc18(PinName rdy, PinName chip_select, PinName int_pin, PinName mosi_pin, PinName miso_pin, PinName sck_pin);
```

Then, you can use the methods provided by the `ADC18` class to interact with the ADC.

Methods

The driver provides several public methods for configuring and reading the ADC.

- `ADC18::ADCData_6Channel getADCData_6Channel()`: Retrieves the 6-channel ADC data.
- `ADC18::ADCData_6Channel getADCData_6Channel_mean(uint8_t samplesToAverage)`: Retrieves `samplesToAverage` samples for each channel of the 6-channel ADC data, then returns the means.
- `int adc18_check_communication()`: Checks the communication with the ADC18 device.
- `int adc18_set_conversion_mode(uint8_t mode)`: Sets the conversion mode of the ADC18 device.
- `void adc18_set_data_rate(uint8_t rate)`: Sets the data rate of the ADC18 device.
- `void adc18_reset_device()`: Resets the ADC18 device.

Private methods include reading the voltage, reading a register, and writing to a register.

Requirements

This driver requires the mbed.h library.

Credits

This project is maintained by Will Stokes.