Firmware Development

- CON-bus
    - Arduino Installation
    - CON-bus Getting Started
    - CON-bus Specification
- Resources / Links
    - Raspbery Pi Pico 2

CON-bus

https://github.com/SoonerRobotics/CON-bus

The goal of CON-bus is to be an easy-to-use on-the-fly parameter configuration library for Arduino-based firmware projects. The CON-bus library includes a CAN driver to allow for CON-bus messages over CAN.

Arduino Installation

Installing CON-bus into the Arduino IDE

Downloading CON-bus

Download the latest Arduino-compatible library `CONBusLib.zip` file at https://github.com/SoonerRobotics/CON-bus/releases/latest

Installing CON-bus

Extract the zip-file to your `Documents\Arduino\libraries` folder, or wherever your Arduino libraries are stored. The final folder structure should look like `Arduino\libraries\CONBusLib` with the `library.properties` file directly in that folder.

CON-bus Getting Started

Getting Started with CON-bus

Loading the example

After following the Arduino Installation instructions, you should see `CON-Bus Lib/CAN_Example` available in the Examples menu under `Examples from Custom Libraries`.

Code Explanation

```
#include <CONBus.h>
#include <CANBusDriver.h>
#include <ACAN2515.h>
```

First, we include all of the necessary dependencies. For the example project, we are using the MCP2515 CAN Controller Driver and so import `ACAN2515.h`.

```
ACAN2515 can(MCP2515_CS, SPI1, MCP2515_INT);


// Setup CONBus variables
CONBus::CONBus conbus;
CONBus::CANBusDriver conbus_can(conbus, 42); // device id 42
```

Here we construct the ACAN2515 object to communicate with the MCP2515 chip. We also construct the main object used by CON-Bus: `CONBus::CONBus`. This object holds all the features of the CONbus library. Finally, we also construct a `CONBus::CANBusDriver` object and pass it the `CONBus::CONBus` object we created earlier. We also pass it a device ID which identifies this specific CONBus device (see the CONbus specifications for more details on CONbus ID).

```
bool useOven = false;
int numberOfOvenRacks = 0;
float ovenTemperatureSetpoint = 100;
```

```
// Create CONBus registers
// The addresses can be anything from 0 to 255
conbus.addRegister(0, &useOven);
conbus.addRegister(4, &numberOfOvenRacks);
conbus.addRegister(21, &ovenTemperatureSetpoint);
```

Here, we create three variables of different types and register them with our CONbus object. By passing the variables by reference to the `addRegister` method, the CONbus library will automatically change the variables without any additional work from the developer.

```
void onCanRecieve() {
  can.isr();
  can.receive(frame);

  conbus_can.readCanMessage(frame.id, frame.data);
}
```

Skipping to the bottom of the example, we can see where we setup `onCanReceive()` which is an interrupt called every time our ACAN2515 driver receives a CAN message. With just one line, we pass the CAN message into our CONbus CAN driver object. The CAN driver will automatically handle checking if the message conforms to the CONbus spec and modifying any variables registerd with CONbus.

```
void loop() {
  if (conbus_can.isReplyReady()) {
    conbus_can.peekReply(outFrame.id, outFrame.len, outFrame.data);

    bool success = can.tryToSend(outFrame);

    // If we successfully send the message, remove the message from the queue
    if (success) {
      conbus_can.popReply();
    }
  }
}
```

Finally, in the Arduino `loop()` method, we continously peek our CONbus CAN driver object to see if there are replies waiting from CONbus to be sent back out over CAN to other devices. For example, these if another device requests a register's value then the CANbus library will generate a reply and store it in the CAN driver until sent. In the above code, we check if a reply is ready, read its value, and attempt to send it out over CAN. If the send was successful, we pop the reply out of the CAN driver so it knows it has been sent.

CON-bus Specification

> WIP: This document is still a Work-In-Progress and only includes the raw CAN definition of CON-bus. In the future, this page should also include more specific details about how CON-bus works and the CAN definition should be under a "CAN Driver Implementation" section.

Configuration (ID 1000-1399)

ID=10xx, Read register, 1 byte

Requests the device with id xx to reply with the value of a register as a specified address. The device will reply with a 11xx message.

Byte 0 is the address of the requested parameter, in the range 0 to 254

- 255 (0xFF) is a special reserved code to receive all parameters

ID=11xx, Read register response, 4-7 bytes

The reply following a 10xx message.

Byte 0 is the address of the parameter being sent Byte 1 is the length in bytes Byte 2 is reserved. Bytes 3-6 is the value of the parameter.

ID=12xx, Write register, 4-7 bytes

Requests the device with id xx to write a value to a parameter with a specified address. The device will reply with a 13xx message.

Byte 0 is the address of the parameter being sent Byte 1 is the length in bytes Byte 2 is reserved. Bytes 3-6 is the value of the parameter to be written.

ID=13xx, Write register response, 4-7 bytes

Response to ID 12xx. Replies back with a copy of the original 12xx message to acknowledge that the parameter was written.

Resources / Links

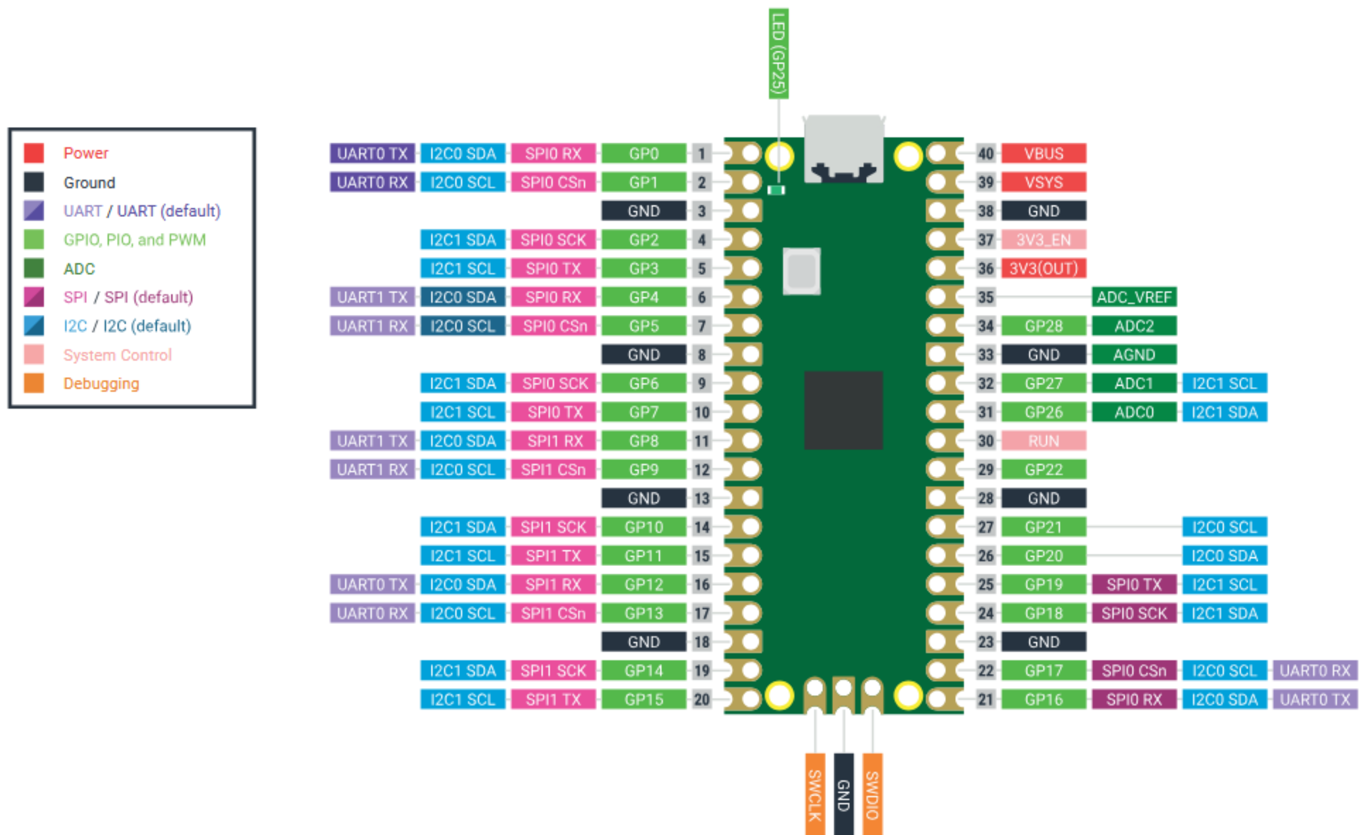
General resources / links that are repeatedly referenced when developing firmware or are just generally helpful.

Raspbery Pi Pico 2

Most PCBs for IGVC and STORM use the Pico 2 as the microcontroller. It has low power draw and a lot of pins, and is also cheap enough that you can burn a few when starting out.

{idk insert other useful information here}

Pinout



(from https://datasheets.raspberrypi.com/pico/Pico-2-Pinout.pdf)

Datasheets

- Pico 2 Datasheet
- This datasheet is more useful for firmware writing/more in-depth: RP2350 Datasheet