Caravel Nucleo Hat

This directory provides a diagnostic software for characterizing timing failure patterns between GPIO pads on Caravel for the MPW-2, MPW-3 and related shuttles.

The diagnostic runs on a STM Nucleo development board in combination with a Caravel Hat board that hosts the Caravel part under test.

The current version of this document can be found at

https://github.com/efabless/caravel_board/blob/main/firmware_vex/nucleo/README.md

or scan the QR code...

Setup

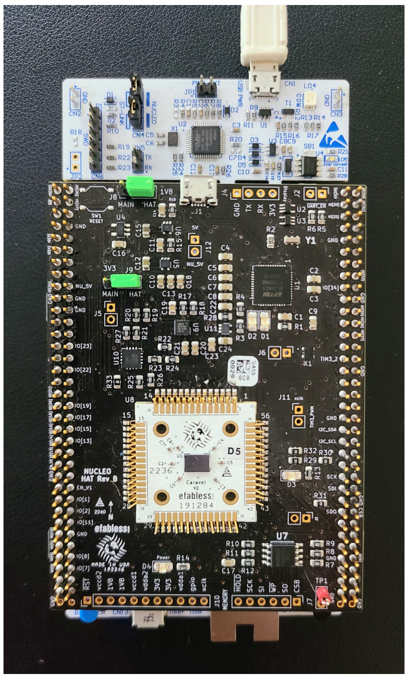
COMPONENTS

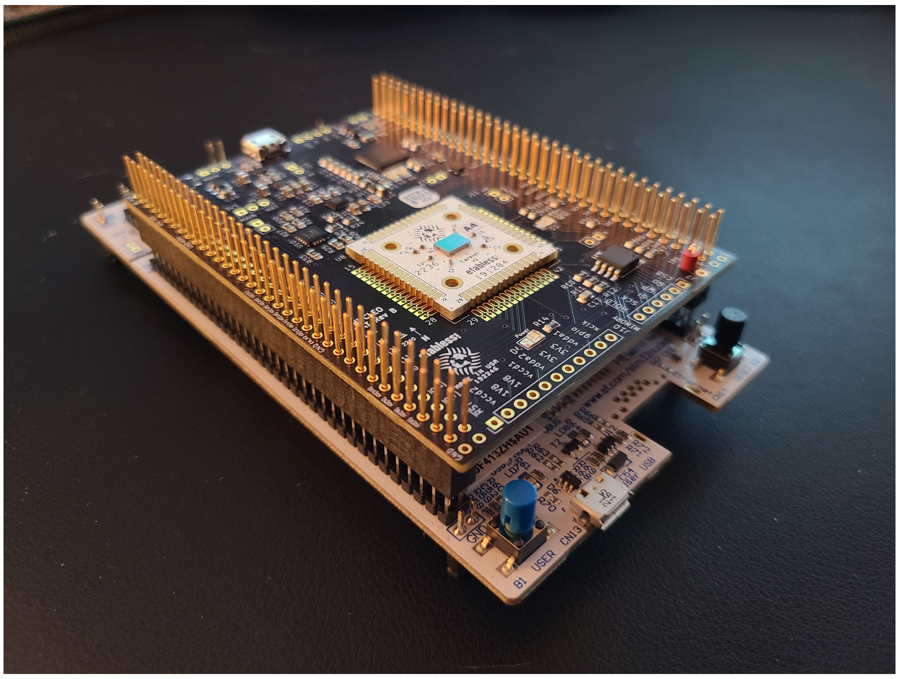
- NUCLEO-F746ZG or NUCLEO-F413ZH
- Caravel Nucleo Hat
- One or more Caravel breakout boards with a Caravel part installed
- Two jumpers for J8 & J9
- USB micro-B to USB-A cable

CONFIGURATION

- Install the jumpers on J8 and J9 in the 'HAT' position to enable the board to be powered by the Nucleo.
- Plug the Caravel Nucleo Hat in Nucleo board pins
  - the USB on the hat should face the ST-LINK breakoff board on Nucleo and away from the push buttons on Nucleo

    - IMPORTANT: the FlexyPin socket allows you to swap breakout boards with different parts. You do not need to solder any pins.
    - Be careful not to bend a pin when inserting the breakout board. If one of the pins bend, use needle-nose pliers to re-straighten it.
- Install a Caravel Breakout board into the socket on the Caravel Hat board
    - the Efabless logo should face the USB connector on the Hat
- Connect the USB cable from the connector CN1 on the Nucleo to a workstation / laptop
- Clone the github repo https://github.com/efabless/caravel_board.git
- Change to the firmware_vex/nucleo directory
- Run `pip install mpremote`





INSTALLATION

```
git clone https://github.com/efabless/caravel_board.git
pip3 install mpremote
```

FINDING YOUR DEVICE

```
mpremote connect list
```

This will verify you can see the Nucleo board through mpremote. The makefile will automatically find and set the device.

RUNNING THE DIAGNOSTIC

```
cd caravel_board/firmware_vex/nucleo
make run PART=<part id>
```

The test will begin with the green LED on the Nucleo flashing 5 times.

When the test concludes, the green and red leds will be as follows:

GREEN	RED	STATUS
2 short + 4 long	off	Full Success - BOTH IO chains configured successfully
2 long	2 short	Partial Success - LOW IO chains configured successfully
4 long	2 short	Partial Success - HIGH IO chains configured successfully
off	2 short + 4 long	Failed - BOTH IO chains failed to configured fully
off	solid	Test failed to complete

If the test completed for the part, run the following to retrieve the configuration file. The file will indicated the IO that were successfully configured. Successfully configured IO can be used for this part for firmware routines.

```
make get_config
```

The file is specific to the part you ran the diagnostic with. Each part will have a different gpio_config_def.py file because the timing failure pattern will be different for each part.

Using the Configuration File

RUN A SANITY CHECK

The following will run a sanity check test using the gpio_config_def.py produced from the diagnostic above. The `gpio_config_def.py` file is stored from the `make get_config` run above and local on your desktop.

To run the sanity check:

```
cd caravel_board/firmware_vex/nucleo
make sanity_check FILE=gpio_config_def.py
```

BUILDING YOUR OWN FIRMWARE

The **gpio_test** directory (`caravel_board/firmware_vex/gpio_test`) provides example for creating your own firmware. We recommend you copy this directory as a template to create your own firmware.

You will need to copy the `gpio_config_def.py` for your part into this directory.

Update `gpio_config_io.c` with the correct IO configuration for your project. Each IO should be set to Management or User mode which defines whether the output is driven from the Management or User area. The IO can be set to output or inputs with either pull-down, pull-up or no terminating resistors.

NOTE: You will not be able to configure any IO that is defined as `H_UNKNOWNED` in your `gpio_config_def.py` file. We recommend setting these IO (as well as any other IO you are not using) to `C_DISABLE` in your `gpio_config_io.py` file.

You can check that your IO configuration to ensure that you can achieve the desired configuration by running `make check` from the project directory. If the configuration can not be configured for that part, you can try changing the configuration or switching to a different part that can me configured.

In your main firmware, you need to include `defs.h` and `gpio_config_io.c` at the top of your file.

Before using IO, you need to call `configure_io()` .

```
#include "../defs.h"
#include "../gpio_config/gpio_config_io.c"

int main() {

  # initialization

  configure_io();

  # my routine using IO pads

}
```

You should leave the Caravel Nucleo board mounted and powered by Nucleo. The timing failure pattern specified in the `gpio_config_def.py` is sensitive to the capacitance loading on the pins that is present when the Caravel board is mounted on the Nucleo and may not be the same when the board is detached.

You can flash and run your firmware with the Caravel Hat mounted on the Nucleo by running:

```
make clean flash_nucleo
```

This will rebuild the firmware prior to flashing Caravel through the Nucleo board. Note, you need to have both USB cables connected to the Nucleo to support this. You also need to have the FLASH variable set correctly per the instructions in the Troubleshooting section below.

Troubleshooting

There are cases where the diagnostic software on the Nucleo may stop working or not work correctly. This is likely due to the Flash filesystem on the Nucleo getting corrupted.

The following steps will re-flash the Nucleo firmware and copy the software to the filesystem.

In addition to **mpremote**, you will need **stlink** and **mpy-cross**

mpy-cross is a cross compiler for micropython the compiles a python file into a binary format which can be run in micropython. It is used here to reduce the size of the files because the size of the flash on the Nucleo board is limited on some models.

To get **mpy-cross**, simply run:

```
pip3 install mpy-cross
```

stlink is a set of utilities for working with the Nucleo boards. We use st-flash to flash a custom micropython image on the board. The customization enables IO on the Nucleo required by the diagnostic for testing Caravel.

Installation for **stlink** is platform specific.

For macOS:

```
brew install stlink
```

For other platforms, see instructions on the github ST-LINK project README...

https://github.com/stlink-org/stlink/tree/master

You will also need to connect both USB ports on the Nucleo to you desktop. The second USB is on the opposite side of Nucleo board from the ST-LINK USB port. This port presents a mountable volume for the Flash filesystem on Nucleo and is how the software and firmware files on copied on to Nucleo. It is also used to retrieve the **gpio_config_def.py** file after the diagnostic completes.

After you made both connections, you will need to find the path for the Flash volume.

On MacOS, it should be located at `//Volumes/PYBFLASH`.

On Ubuntu, it should be mounted at `/media/<userid>/PYBFLASH`.

You will need to `export FLASH=<path>` or set the path in the Makefile at the top of the file.

Once complete, you can re-flash the Nucleo and copy software to the Flash volume by running one of the following make targets based on the model of your Nucleo board. You can find the model of the Nucleo board on a label in the lower left corner of the Nucleo board opposite the ST-LINK breakaway board.

```
# for the F746ZG Nucleo board
make F746ZG

# for F413ZH
make F413ZH
```

You can also just recompile and copy the files onto the flash by running on of the following make targets:

```
# for the F746ZG Nucleo board
make F746ZG-copy

# for F413ZH
make F413ZH-copy
```