EE 492 Project Test Plan

Purpose:

The purpose of this document is to guide the students on how to test Efabless project from our team. This document provides firmware examples, flash programming and diagnostic tools for testing Open MPW and chipIgnite projects using Caravel. It also provides schematics, layout and gerber files for PCB evaluation and breakout boards.

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

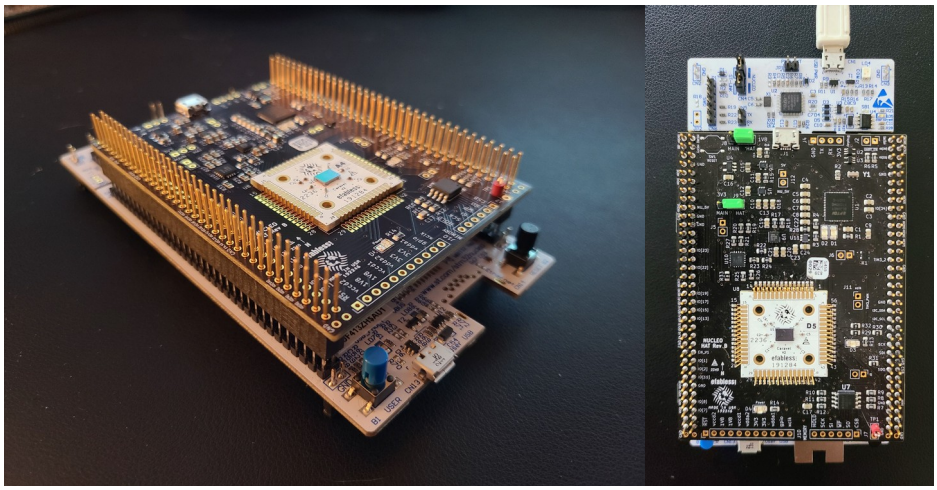

Figure 1: Efabless Caravel Board[1]

Description:

The bottom white board is Caravel Nucleo Board: F746ZG, which provide an affordable and flexible way for users to try out new concepts and build prototypes.

The black board on middle is a custom design from efabless, referred as the "Caravel Nucleo Hat", which provides a diagnostic software for characterizing timing failure patterns between GPIO pads on Caravel for the related shuttles.

The smaller yellow breakout on top (which should be white for most people) is referred as the "Caravel breakout". The fuction of this board is hold our SNN chip for testing.

Prepare:

The purpose of this step is to connect each components together amnd make the boards ready to test. There are some details need to notice during the assembling.

1. Install the jumpers on J8 and J9 in the 'HAT' position to enable the board to be powered by the Nucleo.
2. 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 bends, use needle-nose pliers to re-straighten it.
    - When pressing the Caravel Hat board on the pin headers of the Nucleo, only press far enough to engage the pins. If you press to far, you can short the Flexy pins under the board against jumpers on the Nucleo.
3. Install a Caravel Breakout board into the socket on the Caravel Hat board
    - The Efabless logo should face the USB connector on the Hat

4. Connect the USB cable from the connector CN1 on the Nucleo to your workstation / laptop.
5. Connect a second USB cable from your desktop / workstation from connector CN13 on the opposite side of the Nucleo board from the ST-LINK breakaway board.
    - This port presents a mountable volume for the Flash filesystem on Nucleo and is how the software and firmware files are copied on to Nucleo. It is also used to retrieve the gpio_config_def.py file after the diagnostic completes.

Installation:

The purpose of the installation is to guide the user on how to install and set up the necessary tools required to run a diagnostic software on a customized Micropython image on the Nucleo board. It also provides information on how to connect the Micropython and flash Micropython firmware on the Nucleo board using the stlink tools. Additionally, it provides instructions on how to find the path for the Flash volume on MacOS and Ubuntu, update the diagnostic software, and run the make targets to compile and copy files onto the flash.

1. Install the required tools including mpremote, mpy-cross and rshell. The diagnostic runs on a customized Micropython image on the Nucleo board. The Nucleo firmware image, diagnostic software and Makefile targets for installing and running the routines are located in the firmware_vex/nucleo directory in the caravel_board repo.
    - mpremote is used for connecting the Micropython
    - 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.

2. You will also need to install the stlink tools for your client. These are required to flash Micropython firmware on the Nucleo board.
3. After you made both USB 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.
4. It will be required to update the diagnostic software to get the latest enhancements and bug fixes. 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 (F746ZG or F746ZH). Run one of the following make targets based on the model of your Nucleo board.
5. You can also just recompile and copy the files onto the flash by running on of the following make targets. After the flash completes, check the version of the software.

Test Process:

I. Hardware Test:

Proper hardware testing is essential for ensuring the proper functioning and reliability of a board. The purpose of the hardware testing is to outline some steps that can be taken to test the connectivity, integrity, and functionality of the Caravel Nucleo board's components, as well as to ensure that the power rails have the correct voltage levels. By following these testing plans, any issues with the board can be identified and addressed promptly, leading to better performance and improved user experience.

Input: Check that all pins are connected correctly according to the board schematic. Test each pin individually for connectivity and short circuits

Expected Output: All pins should be connected correctly with no short circuits or open connections.

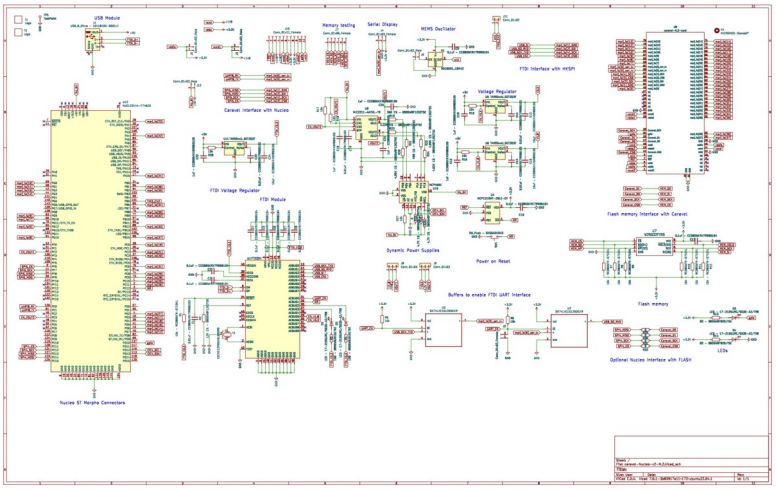

Figure 2: The Schematic of Caravel Nucleo Board[1]

Notice:

- The clock is driven by X1 with a frequency of 10MHz. To drive the clock with custom frequency, disable X1 through J6 and use the external pin for xclk
- The voltage regulator U5 and U6 supply 1.8V and 3.3V through J8 and J9. The traces have to be cut if they are supplied externally.
- vccd1 is connected to 1.8V through J3. The trace has to be cut if it is supplied externally
- vddio is connected to 3.3V through J5. The trace has to be cut if it is supplied externally

- To check if all pins are connected correctly, the following test plan can be developed:
    1. Check the schematic of the board to identify the expected connectivity of the pins.

2. Using a multimeter, verify the connectivity of each pin against the expected schematic connectivity.
3. Verify the connectivity of all power and ground pins.
4. Check the resistance of the pins to identify any potential short circuits or open connections.
5. Verify the signal integrity of the pins by sending test signals and measuring the output.

- To test that all parts of the component on the board are working well, the following test plan can be developed:
    1. Check the datasheet of each component to identify the key parameters that need to be tested.
    2. Develop test cases to verify the correct functioning of each component, based on their respective specifications.
    3. Use appropriate test equipment to measure and record the values of the parameters for each component.
    4. Compare the measured values with the expected values from the datasheet to identify any discrepancies.
    5. Define the acceptable tolerance range for each parameter and establish the pass/fail criteria accordingly.
    6. Conduct additional tests, if necessary, to validate the correct functioning of each component.
- To test that all power rails have the correct voltage levels within the specified tolerance range, the following test plan can be developed:

1. Identify all the power rails on the board.
2. Using a multimeter, measure the voltage levels of each power rail.
3. Compare the measured voltage levels with the expected voltage levels as specified in the datasheet.
4. Define the acceptable tolerance range for the voltage levels based on the specifications.
5. Establish the pass/fail criteria based on the measured voltage levels and the acceptable tolerance range.
6. Conduct additional tests, if necessary, to verify the correct functioning of the power rails.

II. Software Test

The software test provides instructions for setting up and running a diagnostic software that characterizes timing failure patterns between GPIO pads on Caravel for related shuttles. This diagnostic software is designed to run on a customized Micropython image on the Nucleo development board in combination with a Caravel Hatboard that hosts the Caravel part under test. The purpose of this software test is to ensure that the Caravel part under test is functioning correctly by providing detailed instructions on how to run a diagnostic software. The instructions provide guidance on how to use the configuration file, run a sanity check, and build custom firmware. A troubleshooting section is also included to help resolve any issues that may arise during the setup or running of the diagnostic. Running this diagnostic software is an important step in ensuring that the Caravel part under test is functioning correctly and will provide reliable results. For more information, please refer to the Efabless Github.

- To run the diagnostic, enter the following commands. The command depends on whether your project uses Caravel or Caravan (for analog). The PART variable is an ID for the part you are testing defined by you. It will be recorded in the output of the test for future reference.

```
cd caravel_board/firmware_vex/nucleo

## for digital (or analog) projects using Caravel & user_project_wrapper
make run PART=<part id>

## for analog projects using Caravan & user_analog_project_wrapper
make run_analog PART=<part id>
```

Figure 3: Test Code for Caravel

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

Table 1: Test Result Chart

Type Ctrl-C to exit the test diagnostic once it completes. If the test is completed for the part, run the **make get_config** to retrieve the configuration file. The file will indicate the

IO that was successfully configured. Successfully configured IO can be used for this part for firmware routines.

- 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
```

Figure 4: Sanity Check Test

- Before using IO, you need to call configure_io().
- 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.`**
- **GPIO TEST FILE**

  gpio_config_io.py:

  https://github.com/efabless/caravel_board/blob/main/firmware_vex/gpio_test/gpio_config_io.py

gpio_test.c:

https://github.com/efabless/caravel_board/blob/main/firmware_vex/gpio_test/gpio_test.c

io_config.c:

https://github.com/efabless/caravel_board/blob/main/firmware_vex/io_config/io_config.c

III. SNN Chip Test

This is a bring-up test plan for our NMNIST-based Spiking Neural Network (SNN) chip in the Efabless program which is crucial to validate its functionality and performance.

1. **Test objectives:**

- Functional correctness: Ensure that the SNN chip correctly implements the NMNIST-based neural network and can process input data as intended.
- Performance: Validate that the chip meets the target performance metrics (e.g., classification accuracy, latency, power consumption) for the NMNIST dataset.
- Interface compatibility: Confirm that the chip interfaces correctly with external components, such as memory and peripherals.

2. **Test setup:**

- Simulation tools:
  - Icarus Verilog: Used to simulate RTL (Register Transfer Level) designs
  - GTKwave: Used to view the waveform output generated by simulation
- Firmware:
  - Caravel harness repo: Provides makefiles that handle the compilation of CPU software to be run on a simulated CPU along with the rest of the user project wrapper.
- Test scripts:
  - Python: A popular scripting language used for automation and testing.

- JMeter: An open-source load testing tool used for testing the performance and scalability of web applications.
- Postman: A tool used for testing and debugging APIs.

3. **Test cases:**

- Power-on self-test (POST): Verify that the chip initializes correctly upon power-up and performs any necessary self-tests.
- Input/output (I/O) interface tests: Validate the proper functioning of all I/O interfaces, including communication protocols and data transfers with external components.
- Basic functionality tests: Test individual components and sub-blocks of the SNN chip to ensure that they perform as intended (e.g., neuron models, synapses, learning rules).
- Integration tests: Confirm that the SNN chip components work together correctly as a complete system when processing NMNIST data.
- Performance tests: Evaluate the chip's performance against predefined performance metrics (e.g., classification accuracy, latency, power consumption).

4. **Test inputs:**

- Preprocessed NMNIST dataset: Use the NMNIST dataset after applying the necessary preprocessing steps, such as converting images to spike trains or other suitable formats for SNN processing.

5. **Expected outputs:**

- Classification results: The chip should produce correct classification outputs for the input NMNIST dataset with an accuracy comparable to that of the target performance metrics.
- Debug and diagnostic information: Collect relevant debug and diagnostic data during testing to aid in troubleshooting and optimizing the chip's performance.

6. **Test deliverables:**

- Develop a timeline for executing the test plan, including milestones for completing individual test cases and achieving overall objectives
- Test reports: Document the results of each test case, including pass/fail criteria, observed outputs, and any issues encountered.

- Debug and optimization logs: Record any necessary debugging and optimization steps taken during testing, along with their outcomes.

Simulation code:

```
/*
 * SPDX-FileCopyrightText: 2020 Efabless Corporation
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 * SPDX-License-Identifier: Apache-2.0
 */

// This include is relative to $CARAVEL_PATH (see Makefile)
#include <defs.h>
#include <stub.c>

// --------------------------------------------------------

/*
        MPRJ Logic Analyzer Test:
                - Observes counter value through LA probes [31:0]
```

```
		- Sets counter initial value through LA probes [63:32]
		- Flags when counter value exceeds 500 through the management SoC gpio
		- Outputs message to the UART when the test concludes successfuly
*/

void main()
{
	int j;

	/* Set up the housekeeping SPI to be connected internally so	*/
	/* that external pin changes don't affect it.			*/

	// reg_spi_enable = 1;
	// reg_spimaster_cs = 0x00000;

	// reg_spimaster_control = 0x0801;

	// reg_spimaster_control = 0xa002;	// Enable, prescaler = 2,
				// connect to housekeeping SPI

	// Connect the housekeeping SPI to the SPI master
	// so that the CSB line is not left floating.  This allows
	// all of the GPIO pins to be used for user functions.

	// The upper GPIO pins are configured to be output
	// and accessble to the management SoC.
	// Used to flad the start/end of a test
	// The lower GPIO pins are configured to be output
	// and accessible to the user project.  They show
	// the project count value, although this test is
```

```
    // designed to read the project count through the
    // logic analyzer probes.
    // I/O 6 is configured for the UART Tx line

reg_mprj_io_31 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_30 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_29 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_28 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_27 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_26 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_25 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_24 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_23 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_22 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_21 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_20 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_19 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_18 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_17 = GPIO_MODE_MGMT_STD_OUTPUT;
reg_mprj_io_16 = GPIO_MODE_MGMT_STD_OUTPUT;

reg_mprj_io_15 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_14 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_13 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_12 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_11 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_10 = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_9  = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_8  = GPIO_MODE_USER_STD_OUTPUT;
reg_mprj_io_7  = GPIO_MODE_USER_STD_OUTPUT;
```

```
    reg_mprj_io_5  = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_4  = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_3  = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_2  = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_1  = GPIO_MODE_USER_STD_OUTPUT;
    reg_mprj_io_0  = GPIO_MODE_USER_STD_OUTPUT;

    reg_mprj_io_6  = GPIO_MODE_MGMT_STD_OUTPUT;

        // Set UART clock to 64 kbaud (enable before I/O configuration)
        // reg_uart_clkdiv = 625;
        reg_uart_enable = 1;

// Now, apply the configuration
reg_mprj_xfer = 1;
while (reg_mprj_xfer == 1);

// Configure LA probes [31:0], [127:64] as inputs to the cpu
        // Configure LA probes [63:32] as outputs from the cpu
        reg_la0_oenb = reg_la0_iena = 0x00000000;    // [31:0]
        reg_la1_oenb = reg_la1_iena = 0xFFFFFFFF;    // [63:32]
        reg_la2_oenb = reg_la2_iena = 0x00000000;    // [95:64]
        reg_la3_oenb = reg_la3_iena = 0x00000000;    // [127:96]

        // Flag start of the test
        reg_mprj_datal = 0xAB400000;

        // Set Counter value to zero through LA probes [63:32]
        reg_la1_data = 0x00000000;
```

```
        // Configure LA probes from [63:32] as inputs to disable counter write
        reg_la1_oenb = reg_la1_iena = 0x00000000;

        while (1) {
                if (reg_la0_data_in > 0x1F4) {
                        reg_mprj_datal = 0xAB410000;
                        break;
                }
        }
        print("\n");
        print("Monitor: Test 1 Passed\n\n");  // Makes simulation very long!
        reg_mprj_datal = 0xAB510000;
}
```

IV. Reference

[1] Efabless. (n.d.). *Efabless/caravel_board*. GitHub. Retrieved April 17, 2023, from https://github.com/efabless/caravel_board