Bring-Up Plan

Documentation Created By: Aiden Petersen, Matthew Otterson, Regassa Dukele

Information Current as of 11/27/2024

Introduction

The goal of this document is to describe how future students or professors would test the designs that we created. It will contain driver code and potential programs that could be used on the board to correctly use the designs.

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

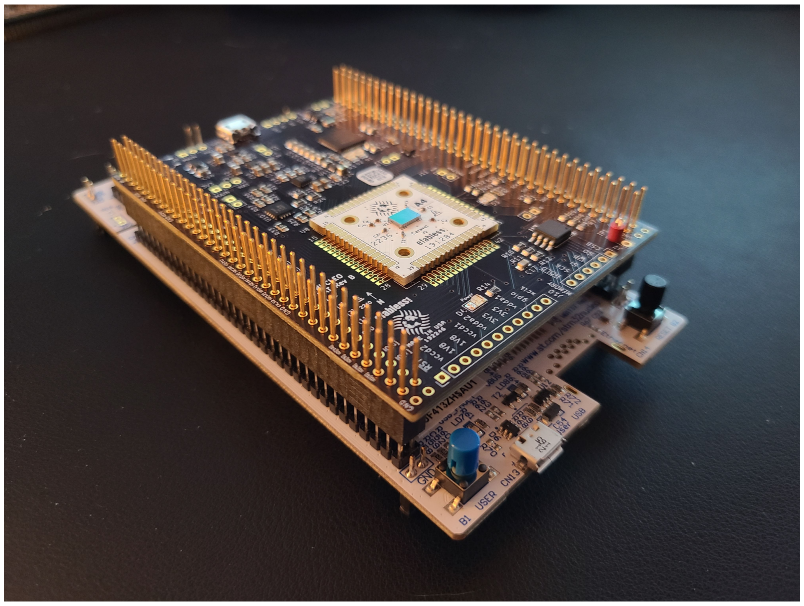

Figure 1: Caravel Nucleo Board [1]

Configuration

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 bend, 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 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 on copied on to Nucleo. It is also used to retrieve the gpio_config_def.py file after the diagnostic completes.

Installation

This will guide the user through installing necessary tools to run diagnostic software through a Micropython image on the Nucleo board.

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.

```
git clone https://github.com/efabless/caravel_board.git
cd caravel_board/firmware/mpw2-5/nucleo
make setup
```

   - **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.

- For macOS:

```
brew install stlink
```

- On Ubuntu download and install a release deb from https://github.com/stlink-org/stlink/releases

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.
    - NOTE: For some linux platforms, the PYBFLASH volume is not automatically installed.

Test Process:

I. Hardware Tests

To ensure the caravel Nucleo board is working correctly a few basic checks should be run on the received board. Compare tests with Schematic of the Caravel Nucleo Board shown Below.

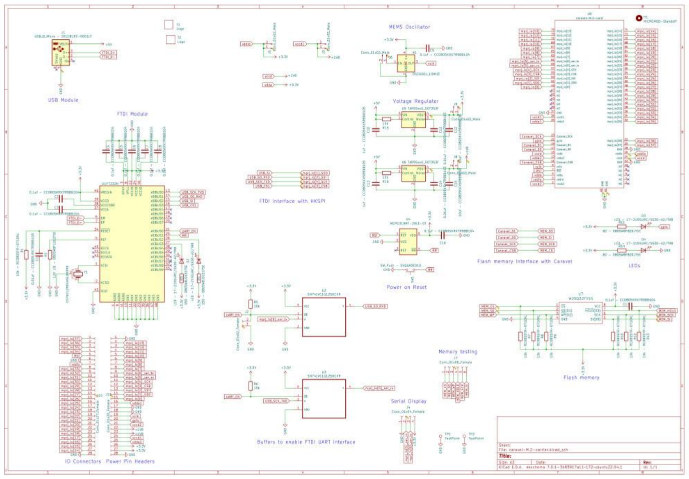

Figure 2: Caravel Nucleo Board [2]

- Use multimeter to verify Expected connectivity compared to schematic by measuring resistances for shorts and Open circuits
- Send and measure test signals through pins to confirm signal integrity.
- Measure voltages on power rails to verify power delivery of pins
    - vccd1 should supply 1.8v
    - vccdio should supply 3.3v

II. Software Tests

This section provides information necessary to setup and run diagnostic software to ensure that the Caravan harness is working properly.

To run the diagnostic, enter the following commands:

```
cd caravel_board/firmware/mpw2-5/nucleo
make run_analog PART=<part id>
```

The test will now run with the green light on the Nucleo flashing 5 times. When the test concludes, the green and red lights 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

Type Ctrl-C to exit the test diagnostic once it completes. You can run

```
make get_config
```

To get a configuration file. The file will indicate if the IO was successfully configured.

III. ReRAM Firmware Test

This is a test plan specific to our ReRAM Crossbar implementation for the Efabless program to validate our crossbar's functionality. The user will need to create test cases using the provided ReRAM crossbar driver APIs. Here is the source code for the APIs that provide a convenient interface with the ReRAM crossbar:

```
#define WAIT_ITERATIONS 10

void write(uint8_t value, uint8_t line){
  uint32_t op;
  // la[ 7: 0]
  uint32_t bitline    = value;
```

```c
  // la[15: 8]
  uint32_t selectline = (0xFF ^ (value)) << 8;
  // la[23:16]
  uint32_t wordline   = (1 << line) << 16;
  uint32_t write_control = (0b11) << 24;

  op = bitline | selectline | wordline | write_control;
  reg_la0_data = op;
  reg_la2_data = op << 19;
  reg_la3_data = op >> 13;
  for(int i = 0; i < WAIT_ITERATIONS; i++){
      __asm("nop");
  }
  reg_la0_data = 0;
}

uint8_t read(uint8_t line){
  uint32_t op;
  uint32_t wordline = (1 << line) << 16;
  op = wordline;
  reg_la0_data = op;
  reg_la2_data = op << 19;
  reg_la3_data = op >> 13;

  for(int i = 0; i < WAIT_ITERATIONS; i++){
      __asm("nop");
  }
  uint8_t result = (uint8_t) reg_la1_data;
  reg_la0_data = 0;
  return result;
}

uint8_t mac(uint8_t value){
  uint32_t op;
  uint32_t selectline = 0;
  uint32_t bitline    = 0;
  uint32_t wordline   = (value) << 16;
  op = selectline | bitline | wordline;
  reg_la0_data = op;
  reg_la2_data = op << 19;
  reg_la3_data = op >> 13;

  for(int i = 0; i < WAIT_ITERATIONS; i++){
      __asm("nop");
  }
  uint8_t result = (uint8_t) reg_la1_data;
  reg_la0_data = 0;
  return result;
}

void form(){
  uint32_t op;
  uint32_t selectline = 0;
  uint32_t bitline    = 0xFF;
  uint32_t wordline   = 0xFF << 8;
  uint32_t form_control = (0b10) << 24;
  op = selectline | bitline | wordline | form_control;
  reg_la0_data = op;
  reg_la2_data = op << 19;
  reg_la3_data = op >> 13;

  uint8_t result = (uint8_t) reg_la1_data;
  return result;
}
```

It exposes 4 functions: write, read, mac, and form. Write will write a value to a line in the crossbar, read will read a line, mac will do a multiply and accumulate operation given a vector value, and form will form all of the ReRAM.

Here is an example of a main function that could use these functions to write an identity matrix, read a row, and then do a multiply and accumulate:

```c
#include <defs.h>
#include <stub.c>
```

```c
void main(){
  // configure logic analyzer
  reg_la0_oenb = reg_la0_iena = 0xFFFFFFFF;
  reg_la1_oenb = reg_la0_iena = 0x00000000;
  reg_la2_oenb = reg_la0_iena = 0xFFFFFFFF;
  reg_la3_oenb = reg_la3_iena = 0xFFFFFF00;

  // Write identity matrix
  write(0x01, 0);
  write(0x02, 1);
  write(0x04, 2);
  write(0x08, 3);
  write(0x10, 4);
  write(0x20, 5);
  write(0x40, 6);
  write(0x80, 7);

  int r = read(3);

  int m = mac(0xFF);

}
```

Expected outputs can be generated using the digital behavioral model. This can be run by following the Digital Model Guide and modifying

`verilog/dv/crossbar_la_test/crossbar_la_test.c`

To match your new test case. You can also use the binary generated by running the test on the digital model to be the firmware on the Nucleo board.

IV. Component Testing Through GPIO:

These tests will allow you to verify the functionality of the components in the user project. Pinout for components in the user project is included at the end of this section.

GPIO and IO pins require external user inputs and readings

8x8 Crossbars

This section will cover the testing of the 0.2V and 0.4V crossbars and their analog control circuits. All tests will be run on both crossbars.

Note: Pin io_analog[5] needs to by supplied -1.8v

The first test to be performed is testing all of the individual 1T1R cells on the crossbars to verify their functionality by writing and checking the High Resistive state and then the Low Resistive State.

1. Perform a write function of 00000000
2. Then Read each cell, the expected output for all cells is 0
3. Perform a write function of 11111111
4. Then Read each cell, the expected output for all cells is 1

Next test the MAC operation, in this test a MAC will be performed

1. Write 00000000
2. Perform MAC 00000000, the expected output is 00000000
3. Perform MAC 11111111, the expected output is 00000000
4. Write 00000001
5. Perform MAC 00000000, the expected output is 00000001
6. Perform MAC 11111111, the expected output is 00000001
7. Repeat 7 times bitshifting the write input left 1 before each Subsequent Test, the expected output will be the same as the write input

The final test to perform on the crossbar is disruption testing. This is to test the effect that performing repeated Read and MAC operations can have on the filament.

1. Write 00000000
2. Perform MAC of 1111111 repeatedly until output is no longer 00000000 or after many attempts with no change
3. Perform Read on all cells to confirm which cell was disrupted
4. Write 11111111
5. Perform MAC of 1111111 repeatedly until output is no longer 11111111 or after many attempts with no change
6. Perform Read on all cells to confirm which cell was disrupted

4x4 Crossbar

There are four 4x4 Crossbars, 2 with normal sized filaments and 2 with large filaments.

To test operations on the ReRAM Cells below are the required voltages

Form	
Bit Line	2.6v - 3.1v
Word Line	1.4v - 2.0v
Select Line	0v

Write	
Bit Line	2.5v / 0v
Word Line	1.8v - 2.5v
Select Line	0v / 2.5v

Read

Bit Line	0.2v - 0.4v
Word Line	1.8v
Select Line	Output

MAC	
Bit Line	0.2v - 0.4v
Word Line	1.8v
Select Line	Output

DAC

There are two one bit DACs included, the expected output is either 0 or 1.8V

Buffer	
Vin	0v / 1.8v
Vout	Output
VDD	1.8v
VSS	0

ADC

There are two one bit ADCs included, the expected output is either 0 or 1.8V, the threshold voltage is ~.59v (R1 = 1943 Ohms, R2 =943 Ohms)

ADC	
Vin	0v - 1.8v
Y	Output

VCC	1.8v
VSS	0

2-1 MUX

The MUXs are analog MUXs so they should output the same input voltages, but there is a loss of current

Buffer	
A	0v / 1.8v
B	0v / 1.8v
S	0v / 1.8v
Vout	S=1 -> A, S=0 -> B
VDD	1.8v
VSS	0

Pinout

Pin	**Connection**
la_data_in [0]	BL 0.2v 1
la_data_in [1]	BL 0.2v 2
la_data_in [2]	BL 0.2v 3
la_data_in [3]	BL 0.2v 4
la_data_in [4]	BL 0.2v 5
la_data_in [5]	BL 0.2v 6
la_data_in [6]	BL 0.2v 7
la_data_in [7]	BL 0.2v 8
la_data_in [8]	SL in 0.2v 1
la_data_in [9]	SL in 0.2v 2
la_data_in [10]	SL in 0.2v 3
la_data_in [11]	SL in 0.2v 4
la_data_in [12]	SL in 0.2v 5
la_data_in [13]	SL in 0.2v 6
la_data_in [14]	SL in 0.2v 7
la_data_in [15]	SL in 0.2v 8
la_data_in [16]	WL in 0.2v 1
la_data_in [17]	WL in 0.2v 2
la_data_in [18]	WL in 0.2v 3
la_data_in [19]	WL in 0.2v 4
la_data_in [20]	WL in 0.2v 5
la_data_in [21]	WL in 0.2v 6

la_data_in [22]	WL in 0.2v 7
la_data_in [23]	WL in 0.2v 8
la_data_in [24]	Write Select 0.2v
la_data_in [25]	Write Form Select 0.2v
la_data_in [83]	BL 0.4v 1
la_data_in [84]	BL 0.4v 2
la_data_in [85]	BL 0.4v 3
la_data_in [86]	BL 0.4v 4
la_data_in [87]	BL 0.4v 5
la_data_in [88]	BL 0.4v 6
la_data_in [89]	BL 0.4v 7
la_data_in [90]	BL 0.4v 8
la_data_in [91]	SL in 0.4v 1
la_data_in [92]	SL in 0.4v 2
la_data_in [93]	SL in 0.4v 3
la_data_in [94]	SL in 0.4v 4
la_data_in [95]	SL in 0.4v 5
la_data_in [96]	SL in 0.4v 6
la_data_in [97]	SL in 0.4v 7
la_data_in [98]	SL in 0.4v 8
la_data_in [99]	WL 0.4v 1
la_data_in [100]	WL 0.4v 2
la_data_in [101]	WL 0.4v 3
la_data_in [102]	WL 0.4v 4
la_data_in [103]	WL 0.4v 5

la_data_in [104]	WL 0.4v 6
la_data_in [105]	WL 0.4v 7
la_data_in [106]	WL 0.4v 8
la_data_in [107]	Write Select 0.4v
la_data_in [108]	Write Form Select 0.4v
la_data_out [32]	SL out 0.2v 1
la_data_out [33]	SL out 0.2v 2
la_data_out [34]	SL out 0.2v 3
la_data_out [35]	SL out 0.2v 4
la_data_out [36]	SL out 0.2v 5
la_data_out [37]	SL out 0.2v 6
la_data_out [38]	SL out 0.2v 7
la_data_out [39]	SL out 0.2v 8
la_data_out [113]	SL out 0.4v 1
la_data_out [114]	SL out 0.4v 2
la_data_out [115]	SL out 0.4v 3
la_data_out [116]	SL out 0.4v 4
la_data_out [117]	SL out 0.4v 5
la_data_out [118]	SL out 0.4v 6
la_data_out [119]	SL out 0.4v 7
la_data_out [120]	SL out 0.4v 8
gpio-analog[0]	4T4R 1 WL 1, 4T4R Large 1 WL 1, Buffer 1 Vin, ADC 1 Vin, MUX 1 A
gpio-analog[1]	4T4R 1 WL 2, 4T4R Large 1 WL 2 , MUX 1 B
gpio-analog[2]	4T4R 1 BL 1, 4T4R Large 1 BL 1

gpio-analog[3]	4T4R 1 BL 2 1, 4T4R Large 1 BL 2
gpio-analog[4]	MUX 1 S
gpio-analog[7]	4T4R 2 WL 1, 4T4R Large 2 WL 1, Buffer 2 Vin, ADC 2 Vin, MUX 2 A
gpio-analog[8]	4T4R 2 WL 2, 4T4R Large 2 WL 2 , MUX 2 B
gpio-analog[9]	4T4R 2 BL 1, 4T4R Large 2 BL 1
gpio-analog[10]	4T4R 2 BL 2, 4T4R Large 2 BL 2
gpio-analog[11]	MUX 2 S
gpio-analog[12]	4T4R 2 SL 1
gpio-analog[13]	4T4R 2 SL 2
gpio-analog[14]	4T4R Large 2 SL 1
gpio-analog[15]	4T4R Large 2 SL 2
gpio-analog[16]	ADC 1 Y
gpio-analog[17]	MUX 1 out
io_analog[0]	4T4R 1 SL 1
io_analog[1]	4T4R 1 SL 2
io_analog[2]	4T4R Large 1 SL 1
io_analog[3]	4T4R Large 1 SL 2
io_analog[5]	VSS Negatvive SL
io_analog[7]	Buffer 1 Vout
io_analog[8]	ADC 2 Y
io_analog[9]	MUX 2 out
io_analog[10]	Buffer 2 Vout

References

[1][2] Efabless. (n.d.). Efabless/caravel_board. GitHub. Retrieved November 29, 2023, from https://github.com/efabless/caravel_board