# Software environment for mannix accelerator

## 1.0 Introduction

The accelerator has software tools. The reasons for that are:

1. Profiling – writing a model in c or other high-level language to find the bottleneck that we want to accelerate.
2. Verification – the system needs to be bit accurate, which mean that output of the accelerator must be equal (byte resolution) to the output of a software program that do the same operation.
3. Modularity – We want the project to be modular and flexible to change. The best way to do so in minimal cost is to manage it with software program.

Those three demands will affect the way the program will be written.

## 2.0 Mannix software manager

Following the introduction, we decided to use this model:

#define ADDRX /\*BASE POINTER OF THE ADDRESS WHERE THE IMAGE WILL BE LOADED \*/

#define ADDRY /\*BASE POINTER OF THE ADDRESS WHERE THE WEIGHTS WILL BE STORED\*/

#define ADDR\_CONV\_RES /\*BASE POINTER OF THE ADDRESS WHERE THE CONVOLUTION LAYER RESULT WILL BE STORED\*/

#define ADDR\_ACTIVATION\_RES /\*BASE POINTER OF THE ADDRESS WHERE THE ACTIVATION LAYER RESULT WILL BE STORED\*/

#define ADDR\_PULL\_RES /\*BASE POINTER OF THE ADDRESS WHERE THE MAX PULL LAYER RESULT WILL BE STORED\*/

#define RESULT /\*\*/

define COUNTER /\*SAVE THE NUMBER OF IMAGE \*/

void MANNIX\_NN(image\_ptr\*, weights\*, bias\*) {

    load(image\_ptr, ADDRX); load(weights, ADDRY); load(bias, ADDRY + B);

MANNIX\_convolution\_layer       (ADDRX,  ADDRY,  ADDR\_CONV\_RES);

MANNIX\_non\_linearity\_activation(ADDR\_CONV\_RES, ADDRY, ADDR\_CONV\_RES);

MANNIX\_pull\_layer              (ADDR\_CONV\_RES, ADDRY, ADDR\_PULL\_RES);

MANNIX\_fully\_conneted          (ADDR\_PULL\_RES, ADDRY, RESULT);

}

2.1 Loading data to the memory:

Since we are working on fpga which is much slower then asic, the first step will be to allocate a memory area to the incoming data.

2.2 MANNIX\_convolution\_layer

The convolution layer. The program must send to the processing unit the starting address of data ( ADDRX), the start address of the weights (ADDRY) and the return address to save the output (ADDRZ). It also must send the data length (Xm), its width (Xn), the window length (Ym) and its width (Yn).

2.3 MANNIX\_pull\_layer

The software must send to the processing unit the start address (ADDRX) and the return address (ADDRZ). It is also must send windows length (Xm), windows width, (Xn),the resulting matrix length (Pm) and the width of the matrix (Pn).

2.4 MANNIX\_non\_linearity\_activation

The unit must send the return address (ADDRZ), this address is also the input address. The software must send the window length (Xm) and window width (Xn). The method of execution will be ReLU.

2.5 MANNIX\_fully\_conneted

The software must send to the processing unit the start address (ADDRX), the weights address (ADDRY), the base address (ADDRB) and the return address (ADDRZ). It is also must send the input vector length (Xn), weight vector length (Ym), weight vector width (Yn) and base length (Bn).

Note: See mannix\_software\_register\_document.doc file.

3.0 software timestamp

The software program will be divided to two parts: pure software, writing to the processing unit.

3.1 Step One - Pure software

Description:

Building software that simulates the behavior of a MANNIX accelerator. The program will fit the software structure mentioned in the introduction. The behavior of the software will be serial, that is - no unit will start work before its predecessor has finished. In addition, each function will simulate the behavior of one of the processing units in such a way that the input and output of every function will be identical to the processing unit which it replaces.

Goal:

* Creating a memory management software shell
* building a model that is easier to test and integrate.
* Modularity.

Expected date: 12/02/21

Notes:

* The model will be written in C in windows operating system in order to facilitate the transition to the RISC-V code later on.
* The chosen dataset is fashion emnist.

3.2 Step Two - Managing a Basic Operating System (Software)

Introduction:

Instead of waiting for its processing to be completed, new data is sent once it is possible. When we come to do this, we encounter two problems:

1. Memory allocation management – for Each image or data that arrives the program must know where it is located.
2. Address Management for Processing Units - Each processing unit handles information independently of the other units. This creates a problem when several images are waiting for the same unit.

The solution to the first problem would be to track an address index. The solution to the second problem would be to manage an address queue for each unit.

Goal:

Creating a memory management model in such a way that it can be replaced by a hardware mechanism.

Notes:

* This step will also be managed in the software only.
* Adding parallel software components (threads) in order to simulate the hardware mechanism. The intention is to create a situation where several units are waiting for the same function.
* Once we have finished processing the image, its place in memory will be vacated.

Expected date: Passover 2021.

* 1. Third stage - integration of processing units within the hardware –

introduction:

in order to use mannix accelerator we need to write to the gpp. We want to create functions that do so and replace the software functions.

Goals:

* Implement each hardware module built into the software system separately.
* Convert program code to RISCV code.

Notes:

* This phase may be parallel to stage 2 depending on the pace of progress of the construction of the processing units.

Expected date: two weeks after Passover.

3.4 Step Four - Create a Python Shell for Code / \* Optional \* /