Dokumentation zum Thema

Binäre Neuronale Netzwerk Beschleuniger

Kevin Woschny(201397)
Somar Iskif(208875)
Michael Krekker(214242)
Thorben Simon(187069)

Kurs: Design of Embedded Systems
Unterthema: Design Your Own CPU
Betreuer: Mikail Yayla
Bearbeitungszeitraum: Juni - 17. September 2021

Inhaltsverzeichnis

1 Einleitung

In dieser Dokumentation wird das Thema BNNs und BNNAs behandelt. Ein 'Binäres Neuronales Netzwerke', kurz 'BNN', ist ein Neuronales Netzwerke, das mit Binären Eingabegrößen arbeitet. Diese Eigenschaft bringt Vorteile im Bezug auf Speicherbedarf und benötigte Rechenleistung. Gerade bei Systemen mit begrenzter Leistung sind diese Eigenschaften gefragt, jedoch verläuft die Überführung ins Binäre nicht ohne Nachteile. Im Punkt Genauigkeit und Training des Neuronalen Netzwerkes stehen Binäre Neuronale Netzwerke etwas schlechter da.
Im Verlauf dieser Dokumentation wird sich ein Binäre Neuronale Netzwerk Beschleuniger genauer angeschaut. Dieses Modul kann die anfallenden Berechnungen eines 'BNN' effizient durchführen. Bei Binären Neuronalen Netzwerken werden Eingabedaten mit einer Gewichts Matrix multipliziert. Bei der Multiplikation macht der Binäre Neuronale Netzwerk Beschleuniger von der Binären Darstellung der Daten Gebrauch. Durch diese Darstellung können Multiplikationen schneller und effizienter durchgeführt werden als Multiplikationen im Integer oder Gleitkommazahl Format.
Im Verlauf des Projekts wurde ein 'BNNA' in 'VHDL' implementiert. Der Aufbau und die Funktionsweise werden im Folgenden erläutert.

2 Processing Unit

Die Processing Unit (PU) verarbeitet die Vektoren die sie vom Controller bekommt. Sie führt also die eigentlichen Berechnungen durch. Wenn sie die Datenvektoren und den Gewichtsvektor fertig verarbeitet hat, vergleicht sie ihr Ergebnis mit dem Treshold. Wenn das Ergebnis über dem Treshold liegt gibt sie 1 aus und sonst 0. Alle nötigen Daten und Signale (data vector, weight vector, treshold, reset signal) erhält die PU vom Controller.
Die Processing Unit besteht aus:

- XNOR
- Popcount
- Register
- Komparator

Abbildung 2.1 zeigt den Aufbau der Processing Unit

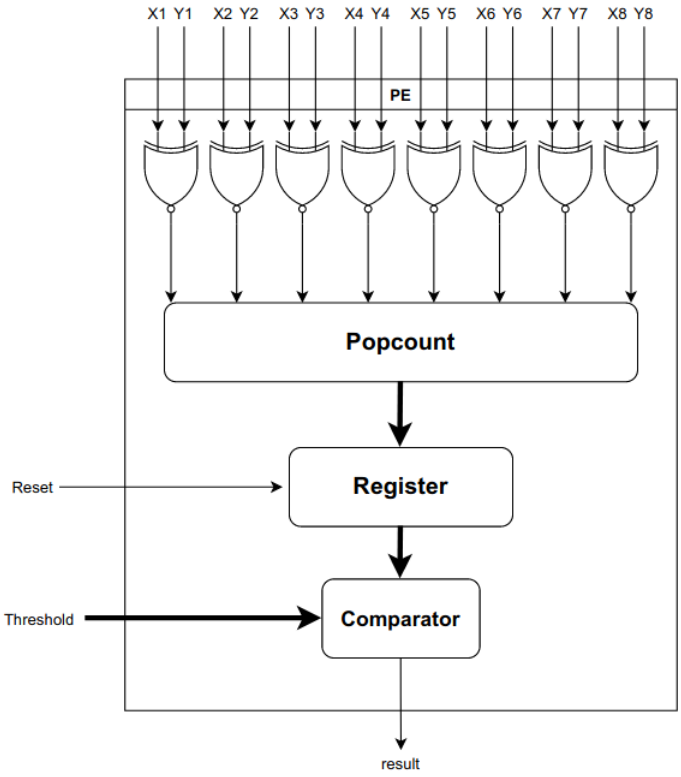

Abbildung 2.1: Aufbau der Processing Unit

2.1 XNOR

Das XNOR Gatter wird verwendet, um zwei Zahlen zu multiplizieren. Da es sich um ein binäres neurales Netz handelt, gibt es nur die Werte -1 oder 1. Intern wird dies mit 0 oder 1 dargestellt. Aus diesem Grund gibt es nur 3 verschieden Möglichkeiten von Multiplikationen. Wenn man nun -1 mit -1 multiplizieren will werden einfach zwei Nullen (denn -1 wird intern mit 0 dargestellt) an die Eingänge eines XNOR Gatters angelegt. Da $0 \sim \oplus 0 = 1$ ist, ist dieses Ergebnis korrekt. Denn es gilt $-1 \times -1 = 1$. Im folgenden sind nochmal alle möglichen multiplikationen dargestellt:

Multiplikation mit XNOR	normale Multiplikation
$0 \sim \oplus 0 = 1$	$-1 \times -1 = 1$
$0 \sim \oplus 1 = 1 \sim \oplus 0 = 0$	$-1 \times 1 = 1 \times -1 = -1$
$1 \sim \oplus 1 = 1$	$1 \times 1 = 1$

Wie man leicht erkennen kann, sind die Ergebnisse korrekt. Um Vektoren zu multiplizieren, kann man einfach mehrere XNOR Gatter parallel schalten. Nun müssen die Ausgänge der XNOR Gatter noch aufsummiert werden, damit man ein korrektes Skalarprodukt erhält. Darum kümmert sich das popcount Modul welches im nächsten Unterkapitel erklärt wird.

2.2 Popcount

Der Popcount zählt die Einsen in einem Bitstream und gibt dann durch die Formel (2N - P) das Ergebnis aus.
Hier ist ein Beispiel, weshalb diese Formel benutzt wird. Wenn z.B. am Popcount-Eingang einen Bitstream anliegt, der wie folgt aussieht: 100111 werden nur die Einsen des Bitstreams gezählt, würde das Ergebnis 4 sein.
Jedoch sollte das Ergebnis 2 sein, da jede Null im Bitstream eine -1 repräsentiert und jede 1 auch eine 1 darstellt. Der Bitstream 100111 steht Stellvertretend für $1 + (-1) + (-1) + 1 + 1 + 1$. Aufaddiert ergibt sich aus dem Bitstream das Ergebnis 2. Dieses Ergebnis gleich dem dem Ergebnis aus der Formel.
Außerdem sind ab dem Output des Popcounts alle Zahlen als Zweierkomplement kodiert um positive wie negative Zahlen einfach summieren zu können.
Abbildung 2.2.2 zeigt den Aufbau des Popcounts

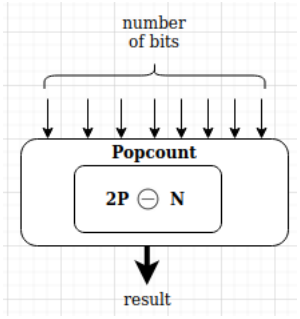

Abbildung 2.2.2: Aufbau des Popcounts

2.3 Register

Das Register akkumuliert die Ergebnisse vom Popcount. Da jede PU nur 8 bit pro Vektor auf einmal verarbeiten kann, müssen größere Vektoren aufgeteilt und stückweise abgearbeitet werden. Das Register summiert somit diese "teil-Skalarprodukte" des Popcounts auf. Wenn der ganze Vektor von der PU abgearbeitet ist und das Register das reset Signal vom Controller bekommt, gibt das Register das Ergebnis aus und setzt seinen Speicher auf 0 zurück.

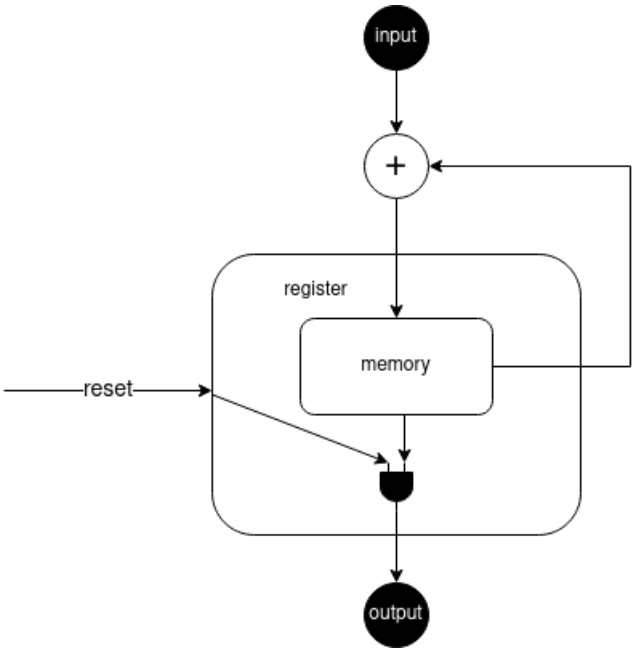

Abbildung 2.3.1: Aufbau des Registers

2.4 Komparator

Der Komparator vergleicht den Threshold mit der Ausgabe von Register. Wenn die Ausgabe vom Register größer gleich dem Threshold ist, dann wird beim Komparator '1' ausgegeben, sonst '0'.
Die Abbildung 2.4.1 zeigt den Aufbau des Komparators.

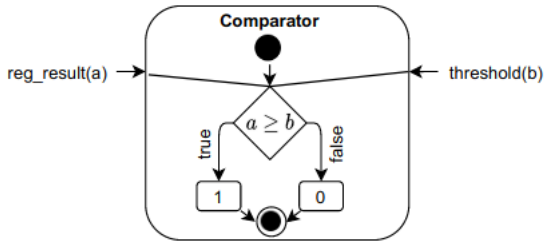

Abbildung 2.4.1: Aufbau des Komparators

3 Controller Unit

Die Controll-Unit steuert den Datenfluss zu den PU´s. Wenn zurzeit keine Berechnungen durchgeführt werden, ist die Controll-Unit bereit Daten anzunehmen. Am 'bnnrdy' Ausgang ist eine logische Eins angelegt, solange keine Berechnungen durchgeführt werden. Über die Eingänge für Input Data ("Data A"), Gewichtsmatrix ("Data B") und Treashholds ("Data T") können pro Takt ein Vektor in die Controll-Unit geschoben werden. Zu jedem Eingang gibt es einen weiteren logischen Eingang, nur wenn dieser Eingang eine Eins anliegen hat, werden die anliegenden Daten abgespeichert, ansonsten werden sie ignoriert. Alle Daten werden von der Controll-Unit abgespeichert. Zum Berechnen müssen zusätzlich noch die Größen der Eingangsdaten übergeben werden. Sobald alles übergeben wurde kann von außen das Startsignal übergeben werden. Mit diesem Signal startet die Berechnungsphase.
Ab diesem Zeitpunkt nimmt die Controll-Unit keine weiteren Daten mehr an. Das "calculator"Modul übernimmt die Befehlsberechnung. Aus den Eingangsgrößen werden die zu bearbeiteten Rechenschritte hergeleitet.
Die Daten müssen geordnet in den BNN geschoben werden. In dieser Reihenfolge werden die Daten auch abgespeichert. Dadurch kann das "calculator"Modul die Befehle erstellen. Aus den Eingangsgrößen kann die Abfolge der nötigen Rechenschritte für Matrixmultiplikation bestimmt werden. Jeden Takt erstellt das "calculator"Modul den nächsten Befehl für einen Rechenschritt. Die Befehle werden in einer 'FIFO' zwischen gespeichert. Die Berechnungen eines Rechenschritts können mehrere Takte in Anspruch nehmen, abhängig von der Länge der Eingangsvektoren, somit Größe der Eingangsdaten. Sollten alle PU´s mit Berechnungen belegt sein, werden keine weiteren Befehle abgearbeitet. In dieser Zeit werden in der 'FIFO' die Befehle zwischengespeichert, damit kein Befehle verloren gehen.
Die eigentliche Abarbeitung der Befehle übernimmt die Controll-Unit. Die Controll-Unit speichert für jede PU den derzeitigen Befehl und den Fortschritt bei diesem Befehl. Pro Takt kann ein Befehl aus der 'FIFO' genommen werden und einer freien PU zugeteilt werden. Jede belegte PU bekommt pro Takt zwei 8-Bit langen Teilvektoren. Sollten die zwei Vektoren, die multipliziert werden, nicht durch 8 teilbar sein, muss im letzten Takt des Rechenschritts aufgefüllt werden. Diese letzten Teilvektoren werden in einem Vektor durch Einsen aufgefüllt und im anderen durch sich abwechselnde Einsen und Nullen aufgefüllt.

3.1 Reset der PU's

Solange eine 'PU' keine Rechnungen durchführt, liegt ein Reset-Bit an. Dadurch wird verhindert, dass die PU ungewollte Rechnungen durchführt, im Register speichert und damit nachfolgende Ergebnisse verfälscht. Während einer Rechnung wird das Reset-Bit der 'PU' auf Null gesetzt und nach der Berechnung wieder auf Eins. Mit dem Reset gibt die 'PU' auch das Ergebnis des Rechenschritts aus, setzt ihr Register auf 0 zurück und ist bereit für die nächsten Rechenschritte. Das setzen der Reset-Bits übernimmt die Controll-Unit. Durch das Speichern des Fortschritts, ist der Controll-Unit bekannt, wann eine 'PU' resetet werden muss. Abhängig davon setzt die Controll-Unit die Reset-Bits der 'PU'.

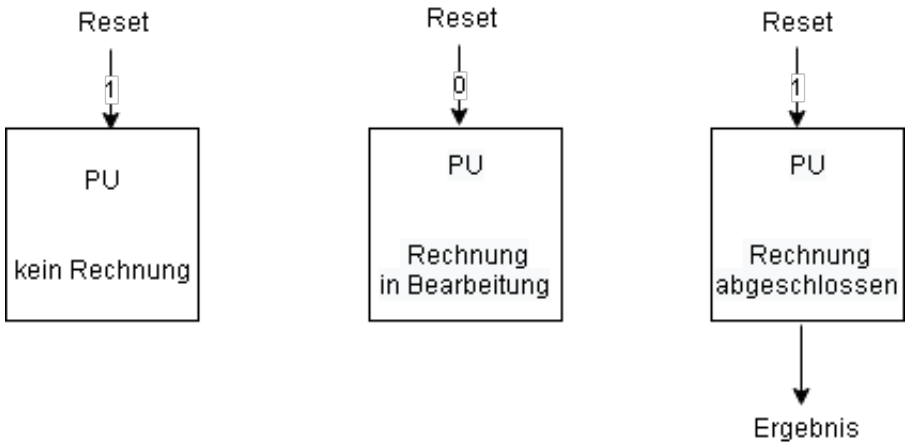

Abbildung 3.1.1: Reset Funktion

4 Output Buffer

Der Output Buffer fängt die Ergebnisse der PUs ein und steuert den Datenfluss hinter den Processing Units. Ein Ergebnis einer PU liegt einen Takt an. Ein Ausgang der PU signalisiert, dass ein Ergebnis ausgegeben wird, der andere Ausgang ist das Ergebnis. In der gleichen Reihenfolge wie die PUs belegt werden, werden die Ergebnisse vom Output Buffer eingesammelt. Hier werden durch die Eingangsgrößen die Größe der Ausgabe bestimmt. Wenn alle Berechnungen durchgeführt worden sind, gibt die Controll-Unit ein Signal. Daraufhin werden die zwischen gespeicherten Ergebnisse ausgegeben und an den Ausgangs des 'BNNA' gelegt. Wenn alle Ergebnisse ausgegeben sind, wird die Controll-Unit benachrichtigt.

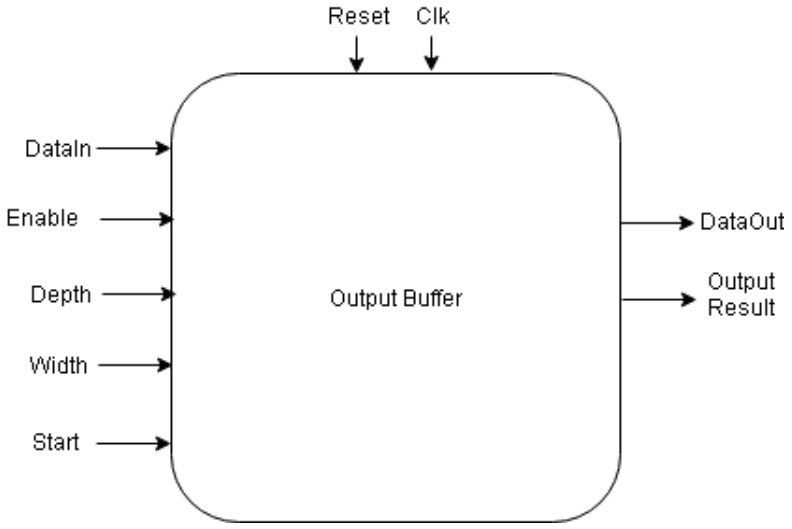

Abbildung 4.1: Aufbau des Output Buffer

5 Input Buffer

Der Input Buffer ist ein eigenständiges Modul und gehört nicht direkt zum 'BNNA'. Mit dem Input Buffer können Daten aus Dateien ausgelesen werden. Diese Daten stammen zum Beispiel aus einem echten 'BNN'. Der Input Buffer berechnet die Dimensionen der Matrix und schiebt die Daten, Zeilen und Spalten, in die richtige Reihenfolge in den 'BNNA' und startet diesen.

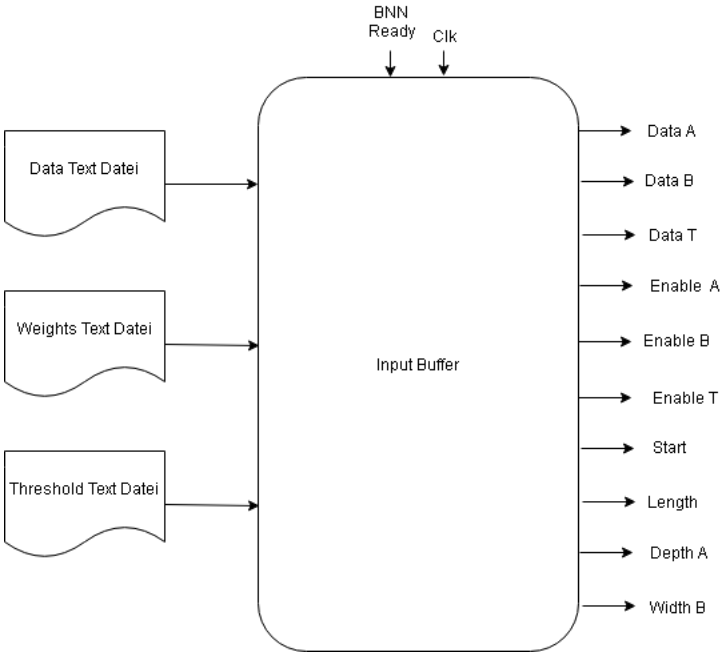

Abbildung 5.1: Aufbau des Input Buffer

6 Fazit

Im verlauf des Projekts wurde ein 'BNNA' im kleinen Maßstab implementiert. Die Implementierung wurde in 'VHDL' vorgenommen. Die Module wurden dort zunächst getrennt von einander implementiert und anschließend zu einem 'BNNA' im kleinen Maßstab zusammen gesetzt. In Zukunft denkbar wären das 'BNNA' in einen größeren Maßstab zu skalieren oder das 'BNNA' auf einem 'FPGA' laufen zu lassen.
Durch die Implementierung in kleinen Modulen, kam es beim Zusammensetzten zu Problemen. Die Informationslage zum 'BNNA' Aufbau ist recht dünn, weshalb es beim Implementieren der Module viel Spielraum gab. Dadurch gab es Probleme bei der Kommunikation zwischen den Modulen. Die Kommunikation untereinander im Online Format kann noch verbessert werden, um solche Problem zu vermeiden. Dem Online Format geschuldet gab es Schwierigkeiten gemeinsam am gleichen Problem zu arbeiten und schnell Lösungen für kleine Probleme zu finden, sodass diese manchmal viel Zeit beanspruchten.
Trotz alledem konnte gezeigt werden, dass der 'BNNA' die Binäre Darstellung zur einfachen Berechnung nutzten kann, um eine Multiplikation mit Hilfe der Elementen der Processing Unite durchzuführen.