Deutsches Zentrum
für Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft

Diplomarbeit

Entwurf eines RCE-Netzwerk-Klassifikators in konfigurierbarer Hardware

Vorgelegt von:
Emanuel Schlüßler

Königs Wusterhausen, Oktober 2007

Betreut von :
Prof. Dr. Helmut Jürgensen (Uni Potsdam)
Dr. David Krutz (DLR)

Diplomarbeit am
Institut für Robotik und Mechatronik
Einrichtung Optische Informationssysteme
Deutsches Zentrum für Luft- und Raumfahrt (DLR)
Thema : Entwurf eines RCE-Netzwerk-Klassifikators in konfigurierbarer Hardware
Autor : Emanuel Schlüßler

Emanuel Schlüßler
Clara-Schumann-Str. 3
15711 Königs Wusterhausen
email: Emanuel.Schluessler@gmx.de

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, die vorliegende Diplomarbeit selbständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Königs Wusterhausen, 19. Oktober 2007 _____

Schlüßler, Emanuel

Danksagung

An dieser Stelle möchte ich zunächst meiner Familie, vor allem meinem Vater Werner Schlüßler und meinen Großeltern Edith und Wolfgang Schlüßler danken, die mich auf dem langen Weg, wo sie nur konnten, unterstützt haben und ohne die vieles nicht möglich gewesen wäre.

Ebenso danke ich meiner lieben Frau Katrin, die stets an meiner Seite stand und mir immer den Rücken gestärkt und wenn nötig freigehalten hat.

Außerdem gilt Dr. David Krutz, Dr. Winfried Halle und nicht zuletzt Prof. Dr. Jürgensen mein Dank, deren Bereitschaft es geschuldet ist, dass ich mich genau diesem Thema widmen durfte. Davids Hilfe bei der Klärung vieler Sachverhalte hat mir so manche Tür geöffnet und die Qualität der Arbeit ständig voran getrieben.

Danken möchte ich auch meinem Wegbegleiter Patrick Scherbaum, nicht nur auf dem Weg zum Diplom, sondern auch auf den kraftgebenden Wegen zum Kaffeeautomaten.

Zusammenfassung

Bei den RCE-Netzwerken handelt es sich um Klassifikatoren, die multimodale Klassenverteilungen erfassen können und unregelmäßige Klassenstrukturen im Merkmalsraum erlauben. Trotz dieser Vorteile kommen sie weitestgehend mit einfachen Operationen aus, so dass eine Implementierung in Hardware naheliegt. Der Neurochip NI-1000 realisiert diese Verfahren zur Klassifikation hochdimensionaler Muster. Er hat jedoch den Nachteil, die Merkmale nur mit fünf Bit aufzulösen. Um die Vorteile der RCE-Netzwerk-Klassifikation ohne diese Beschränkung nutzen zu können, bietet sich ein Hardwareentwurf für FPGAs an, der leicht in seiner Auflösung variiert werden kann. Diese Arbeit beschreibt den Entwurf der RCE-Netzwerk-Klassifikatoren in Hardware. Es wird eine Erklärung dafür geliefert, warum die RCE-Netzwerke besonders gut geeignet sind und welche Realisierungsvarianten bei der Umsetzung in Frage kommen. Eine Leistunganalyse stellt die Resultate den Fähigkeiten des NI1000-Neurochips gegenüber, um die Ergebnisse zu bewerten.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Mit fortschreitender Automatisierung technischer und elektronischer Prozesse wuchs das Bedürfnis, prozesssteuernde Entscheidungen durch Rechner und Maschinen realisieren zu lassen. Erst dadurch wird dem Wunsch nach vollautomatischer Prozessdatenverarbeitung und Steuerung Genüge getan. Noch heute müssen viele Entscheidungen, die Prozessabläufe steuern, von Menschen getroffen werden, da die richtige Lösung oft nur durch die hohen kognitiven Fähigkeiten des menschlichen Gehirns möglich werden. Daher wird vielfältig versucht, die automatische Entscheidungsfindung an die menschliche Leistung anzupassen. Nur dann kann von einer gelungenen Lösungsvariante gesprochen werden.

Für den Menschen ist es kein Problem, auf dem Foto einer Menschenmenge denjenigen mit dem roten Hut zu finden, während eine Maschine hierfür Erhebliches leisten muss. Gegebene Daten, die oft unstrukturiert sind und in einer technischen, maschinennahen Form vorliegen, fehlt es zunächst an jeglichem Kontext. Es sind daher Verfahren nötig, die aus diesen Daten die gewünschte Information extrahieren und effizient darstellen.

Die Mustererkennung ist eine Teildisziplin der Signalverarbeitung und verfolgt das Ziel, in Signalen oder Daten auftretende Muster bestimmten Klassen zuzuordnen. Solche Klassen repräsentieren viele Muster und bilden eine Generalisierung[1]. Genau diese Fähigkeit ermöglicht einem Menschen, Tische von Stühlen und Äpfel von Birnen zu

[1] Generalisierung oder Abstraktion bezeichnet die Verallgemeinerung von konkreten Objekten zu einer Klasse von Objekten.

unterscheiden. Jeder Tisch hat bestimmte Eigenschaften, die ihn deutlich von Stühlen unterscheiden.

Ein Klassifikator dient bei diesem Vorgang als Zuordner, der ein Muster einer oder mehreren Klassen direkt oder mit einer bestimmten Wahrscheinlichkeit zuordnet. Neben den klassischen Verfahren haben sich Klassifikatoren auf Basis künstlich-neuronaler Netze als geeignet erwiesen.

Weitaus häufiger als herkömmliche Computer werden Speziallösungen wie Mikrocomputer, DSPs (engl. digital signal processor) oder Neurochips zur Signalverarbeitung eingesetzt. Sie genügen meistens nicht dem Von-Neumann-Modell, sind aber besser an die jeweiligen Anforderungen angepasst. Dadurch werden die heute oft nötigen hohen Datenraten in der Echtzeit-Signalverarbeitung erst ermöglicht.

1986 brachte die Firma Xilinx einen Halbleiterbaustein auf den Markt, der die Fähigkeit besaß, neu konfiguriert (programmiert) werden zu können. Mit diesen Chips kann jede logische Schaltung realisiert werden,vom TV-Decoder bis zum Prozessor. Diese Field Programmable Gate Arrays (FPGAs) haben inzwischen einen hohen Verbreitungsgrad erlangt, da sie zum einen die Vorteile von Hardware mit sich bringen und zum anderen frei und immer wieder programmiert werden können. Hierfür stehen Hochsprachen wie VHDL oder Verilog zur Verfügung, die ürsprünglich zur reinen Simulation von Schaltungen entwickelt wurden.

1.1 Motivation

Der Neurochip NI1000 ist eine kommerzielle Speziallösung und realisiert RCE- und PRCE-Netzwerke zur Klassifikation von hochdimensionalen Mustern. Es handelt sich dabei um einen sogenannten Multireferenzklassifikator, der multimodale Klassenverteilungen (Klassen mit mehreren Häufungspunkten) erfassen kann. Nachteil des NI1000 ist die geringe Auflösung der Merkmale mit 5 Bit und die Beschränkung auf 64 Klassen, zwischen denen höchstens separiert werden kann.

FPGAs sind das geignete Mittel, die Funktionalität des NI1000 ohne die genannten Beschränkungen umzusetzen. VHDL-Programme können problemspezifisch mit geeigneter Bitbreite und der richtigen Anzahl von Klassen in ein Hardwarelayout übersetzt werden. Somit ist die Größe der Schaltung an das Problem angepasst und evtl. auf dem Chip verbleibender Platz kann für andere Aufgaben genutzt werden.

Wesentliche Triebfeder für diese Arbeit ist die Feuererkennung des Kleinstsatelliten *BIRD* (Bi-spectral Infra-Red Detection)[LBB+04]. Hier kommt der Neurochip NI1000 zum Einsatz, dessen Genauigkeit durch die Bitbreite von fünf Bit bei der Merkmalsauflösung beschränkt ist. Die Technologie des BIRD wird im Moment weiterentwickelt

und dient als Grundlage für zukünftige Satellitenprojekte zur Feuererkennung beim Deutschen Zentrum für Luft- und Raumfahrt (DLR).

1.2 Ziel dieser Arbeit

Mit dieser Arbeit soll die Klassifizierung nach dem RCE-Netzwerk in VHDL realisiert werden. Der nötige Trainingsalgorithmus, der das Netzwerk auf die Klassifizierung vorbereitet, wird durch ein Softwaremodul bereitgestellt und nicht in Hardware realisiert. Das wird vor allem dadurch begründet, dass die Struktur der Schaltung von den Trainingsdaten abhängen kann.

Vor der Umsetzung werden verschiedene Möglichkeiten der Realisierung zu diskutieren sein, die sich in Platz- und Zeitbedarf unterscheiden. Anforderungen und technische Mittel entscheiden letztlich, welche Variante zu bevorzugen ist.

1.3 Gliederung

Kapitel 2 liefert eine kurze Einführung in die Signalverarbeitung und Mustererkennung und schafft einen Überblick über Klassifikatoren, wodurch diese Arbeit und der Klassifikator eingeordnet werden kann.

In Abschnitt 2.2.2 werden verschiedene Klassifikatoren gegenübergestellt. Es soll gezeigt werden, warum die RCE-Netzwerke als Klassifikatoren besonders geeignet sind, in Hardware implementiert zu werden. Die genauen Details der RCE-Netzwerk-Klassifikation und des Trainingsalgorithmus werden in Kapitel 3 vorgestellt.

Mit dem Ziel einer Hardwareimplementierung ist auch immer die Frage nach Zeit und Platzbedarf einer Schaltung verbunden. Für jede Aufgabenstellung existieren verschiedene Varianten der Realisierung, die jeweils einen Kompromiss zwischen Zeitverhalten und Platzbedarf darstellen. Kapitel 4 widmet sich diesem Thema und zeigt einige der Ansätze, die sich deutlich unter diesen Gesichtspunkten unterscheiden. Aus diesen Überlegungen heraus wird eine Entscheidung über das Layout der Schaltung zu treffen sein, welche Ausgangspunkt für die Implementierung sein wird.

Auf Grundlage der Kapitel 2 bis 4 kann der Klassifikator nun implementiert werden. Die resultierende Architektur wird in Kapitel 5 vorgestellt. Dabei wird ausgehend vom Klassifikator als Modul seine Schnittstelle beschrieben und anschließend nach dem Top-Down-Prinzip seine wesentlichen Komponenten.

Nach der Diskussion zu möglichen Realsierungsvarianten in Kapitel 4 und durchgeführter Implementierung stellt sich die Frage nach dem tatsächlichen Zeit- und Platz-

verhalten der Schaltung. Da die Schaltung nach dem Pipeline-Prinzip umgesetzt wird und die Latenz für den praktischen Einsatz von Bedeutung ist, wird in Kapitel 6 die Schaltung unter dieser Fragestellung analysiert.

Der Trainingsalgorithmus, der den Klassifikator konfiguriert, wird nicht in Hardware implementiert, sondern durch ein Softwaremodul bereitgestellt. Diesem Thema widmet sich Kapitel 8. Es werden vor allem algorithmische Besonderheiten aufgezeigt, die von den Ausführungen in Abschnitt 3.1 abweichen.

Abschließend folgt eine Zusammenfassung der erreichten Ergebnisse sowie eine Bewertung der Resultate.

2 Grundlagen

Kapitel 2 widmet sich den Grundlagen dieser Arbeit. Zunächst werden allgemeine Aufgaben der Signalverarbeitung vorgestellt, bevor die Bedeutung von Klassifikatoren für die Mustererkennung veranschaulicht wird. Neben den RCE-Netzwerken gibt es eine Vielzahl weiterer Klassifikatoren, von denen einige vorgestellt werden. Dadurch soll die Bedeutung der RCE-Netzwerke und ihre besondere Leistung ersichtlich werden. Einen kurzen Einblick in das Thema *Field Programmable Gate Arrays* liefert Abschnitt 2.3. Diese Arbeit verfolgt nicht das Ziel, FPGA-Technologien zu vergleichen oder umfassend vorzustellen. Vielmehr soll in Abschnitt 2.3 einen Eindruck vermittelt werden, welches Potential in diesen Bausteinen steckt.

2.1 Signalverarbeitung

Signalverarbeitende Prozesse erfordern fast immer, dass Information aus einem Informationsträger gewonnen und anschließend verarbeitet wird. Das Resultat wird abschließend wieder mit einem Informationsträger nach außen sichtbar gemacht. Das hier erkennbare EVA-Prinzip[1] - ein Grundschema der elektronischen Datenverarbeitung - bildet ein abstraktes Modell, das in der heutigen Zeit unzählige Realisierungen findet. Es beschreibt in fundamentaler Weise alle elektronischen Prozesse.

[1] Eingabe-Verarbeitung-Ausgabe

Ein solches informationsverarbeitendes System verfügt demnach über ein Eingangssignal $f(t)$, das mit einer Vorschrift F in ein Ausgangssignal $g(t) = F\{f(t)\}$ transformiert wird.

Für den Fall, dass die Information in einer physikalischen Variablen wie Temperatur, Druck oder Spannung steckt, spricht man von analogen Signalen. Die Verwendung des Begriffes *analog* unterstreicht die Tatsache, dass sie einen überabzählbar großen Wertebereich haben. Fast immer kommen Digitalrechner bei der Verarbeitung zum Einsatz. Daher müssen die Signale digitalisiert werden, wodurch ihr Wertebereich abzählbar und somit für Rechner handhabbar wird. Man spricht dann von digitalen Signalen.

Die wissenschaftliche Disziplin, die sich mit der Verarbeitung von Signalen beschäftigt, ist die Signalverarbeitung, deren Aufgaben sich zusammenfassend mit den drei Hauptzielen Informationsgewinnung, Signalverbesserung und Signalkompression beschreiben lassen [MH04]:

Informationsgewinnung Verfahren der Signalverarbeitung werden eingesetzt, um Information über den signalerzeugenden Prozess zu gewinnen oder um seinen Zustand zu beschreiben.

Signalverbesserung Verfahren, die die im Signal enthaltene Nutzlast hervorheben, bzw. irrelevante und störende Anteile eleminieren. Verbessern kann auch bedeuten, dass ein Signal nur in der Art und Weise aufbereitet wird, dass es durch weiterverarbeitende Prozesse besser gelesen oder interpretiert werden kann.

Signalkompression Darunter versteht man i.A. die Verdichtung der Signale, ohne die wesentliche Information zu verlieren. Ohne Kompression können die heute nötigen hohen Durchsatzraten nur selten erreicht werden.

Der RCE-Klassifikator ist ein Verfahren der Informationsgewinnung und Kompression. Für eine umfassende Einführung in die Signalverarbeitung sei auf [MH04] und [SH99] verwiesen. Die genannten Ziele verweisen lediglich auf die Hauptanliegen der Signalverarbeitung, lassen aber viele Schritte der Vor- und Nachbereitung außer Acht. Das zu Beginn des Kapitels erwähnte EVA-Prinzip findet in der Signalverarbeitung eine Entsprechung - die Signalverarbeitungskette - und soll im Folgenden kurz vorgestellt werden. Dadurch kann diese Arbeit und der Klassifikator besser eingeordnet werden.

2.1.1 Aufbau einer Signalverarbeitungskette

Die Signalverarbeitungskette (siehe Abb. 2.1) ist ein idealisiertes Modell, das den Informationsfluss von einer nichtelektrischen/analogen Quelle hin zu einer nichtelektrischen/analogen Senke beschreibt. Die Verarbeitung mit Digitalrechnern erfordert jedoch digitale Signale. Daher müssen die nichtelektrischen Signale zunächst in elek-

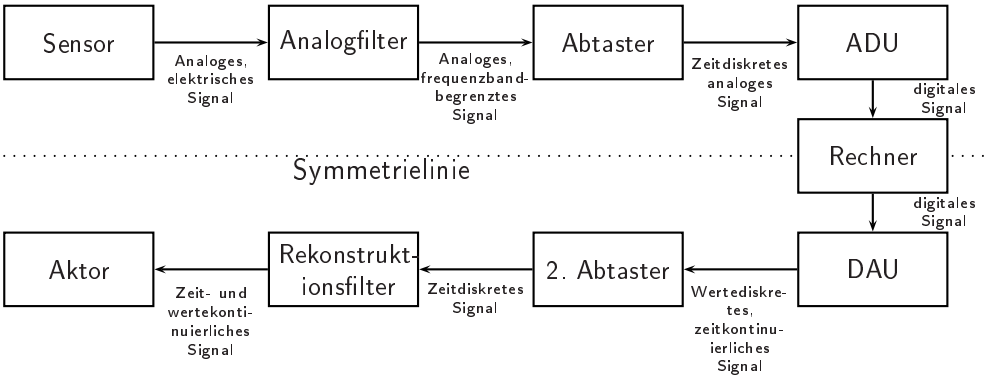


Abbildung 2.1: Signalverarbeitungskette

trische (Sensor) und dann in digitale Signale (Abtaster/ADU) transformiert werden. Nach Abschluss der Signalverarbeitung durch einen Digitalrechner wird das Signal zunächst wieder in ein elektrisches verwandelt (DAU/Abtaster/Filter), bevor ein Aktor ein nichtelektrisches erzeugt. Für nähere Ausführungen zu den einzelnen Komponenten sei wieder auf [MH04] und [SH99] verwiesen.

Ein Klassifikator ist wie jeder andere Signalverarbeitungsoperator mit digitalem Ein- und Ausgang eine datenverarbeitende Maschine, die in der Signalverarbeitungskette dem Rechner zuzuordnen ist.

Die Behauptung, dass Ausgangsgrößen der Signalverarbeitung nichtelektrisch sind, scheint zunächst verwunderlich. Dieser Punkt wird klarer, wenn die Frage gestellt wird, für welche Reize der Mensch Sinnesorgane hat. Sieht man von der unangenehmen Wirkung eines elektrischen Schlages ab, so verfügt der Mensch über keinen Elektrizitätssinn. Auch bei technischen Prozessen werden Aktoren eingesetzt, die die Information in physikalisch wirkende Kräfte wandeln.

2.1.2 Zusammenfassung

Ein Klassifikator ist ein signalverarbeitender Operator, der darüber hinaus der Mustererkennung (siehe 2.2) zuzuordnen ist. Nach obiger Schreibweise wird ein Muster $f(t)$ durch einen Klassifikator F einer Klasse $g(t) = F\{f(t)\}$ zugeordnet. Es sei zu bemerken, dass die Verwendung von t als Parameter eine von der Zeit abhängige Funktion vermuten lässt. Viele physikalische Prozesse treten tatsächlich in dieser Form auf und werden auch als Zeitfunktionen bezeichnet.

2.2 Mustererkennung

Die Mustererkennung ist ein Teilgebiet der Informatik und bezeichnet das maschinelle Erkennen und Auswerten von Mustern in Signalen. Erkennen bedeutet hier, ein Muster wegen seiner Lage im Merkmalsraum einer bestimmten Klasse zuzuorden. Die Gesamtheit der Merkmale, die ein Objekt charakterisieren und dessen Lage im Merkmalsraum bestimmen, wird Muster genannt. Ein D-dimensionaler Merkmalsraum ist ein Vektorraum, wobei D die Anzahl der zu untersuchenden Merkmale ist.

Von den realen Objekten der Umwelt muss so abstrahiert werden, dass die entstehende Repräsentation von Digitalrechnern verarbeitet werden kann. Dazu werden aus Sensordaten $\{v_1, v_2, ..., v_i\}$ die interessanten Merkmale $\{x_1, ..., x_j\}$ extrahiert (Merkmalsextraktion) und zu einem Merkmalsvektor $\vec{x}$ zusammengefasst. Es kann vorkommen, dass bestimmte Merkmale überflüssig sind, weil deren Bedeutung bereits durch andere Merkmale erfasst wird oder sie keine Aussage bezüglich der gewünschten Klassifikation treffen können. Im Schritt der Merkmalsreduktion wird versucht, die wesentlichen Merkmale zu finden und redundante zu streichen, wodurch die Dimension des Merkmalsraums verringert wird. Dieser Schritt findet vor der eigentlichen Klassifikation statt und soll helfen, den Leistungsaufwand für die gesamte Signalverarbeitung zu reduzieren.Der Vektor $\vec{x}$ wird als Muster bezeichnet und verweist auf einen bestimmten Punkt im D-dimensionalen Merkmalsraum.

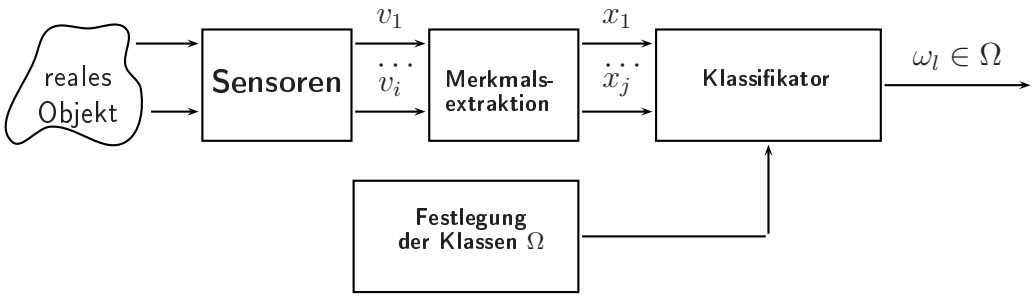


Abbildung 2.2: Mustererkennung

Anhand bestimmter Bedingungen, wie Diskriminanzfunktionen oder Trainingsdaten, wird der Merkmalsraum in eine Menge Ω von Regionen unterteilt, welche den einzelnen Klassen $\{\omega_1, \omega_2, ...\omega_k\}$ entsprechen. Fällt ein Muster in eine der Regionen $\omega_i \in \Omega$, so wird das Muster dieser Klasse zugeordnet und als ein Objekt dieser Klasse interpretiert.

2.2.1 Systematisierung von Klassifikationsverfahren

Heutzutage existieren Klassifikatoren unterschiedlichster Art und es ist immer nötig, den geigneten für eine bestimmte Klassifikationsaufgabe zu finden. Eine umfassende Einführung in die Mustererkennung kann in [DSH00] nachgelesen werden. Die folgende Übersicht liefert eine Gegenüberstellung verschiedener Eigenschaften, nach denen sich Klassifikatoren unterscheiden lassen. Die Auswahl wurde so getroffen, dass der RCE-Klassifikator gut eingeordnet werden kann. Einen Anspruch auf Vollständigkeit erhebt die Aufzählung nicht.

Statistische vs. verteilungsfreie Verfahren: Merkmalsräume verteilungsfreier Verfahren sind in klar abgegrenzte Zonen unterteilt, wogegen statistische Verfahren von Wahrscheinlichkeiten oder Dichteverteilungen Gebrauch machen. In letzterem Fall wird nicht eindeutig, sondern mit einer gewissen Wahrscheinlichkeit die Zugehörigkeit zu einer Klasse angegeben.

Überwachte vs. unüberwachte Verfahren: Liegt der Konfiguration des Klassifikators eine Menge von Lerndaten zu Grunde, so spricht man von überwachten Verfahren. Die Lernstichprobe ist eine Menge von Mustern, deren Klassenzuordnung bekannt ist. Dadurch ist eine Einteilung des Merkmalsraums vor der Klassifikation möglich, jedoch ändern sich Lage und Anzahl der Klassen während der Klassifikation nicht. Unüberwachte Verfahren, wie zum Beispiel Clustering-Methoden, werden vor der Klassifizierung nicht trainiert. Die Aufteilung des Merkmalsraums ergibt sich während der Klassifikation, je nachdem wie sich die gemessenen Muster in Lage und Anzahl über den Merkmalsraum verteilen. Da sich die Aufteilung der Merkmalsräume völlig autonom einstellt, wird auch von unüberwachten Verfahren gesprochen.

Parametrische vs. nichtparametrische Verfahren: Beruht die Klassenbeschreibung auf einer Verteilungsfunktion, kann sie über Parameter eingestellt werden. Es gibt aber Klassen, deren Verteilungsdichte auf diese Weise nicht modelliert werden kann. Man spricht dann von nichtparametrischen Klassifikatoren, denen keine Einschränkungen an die Form der Verteilungsdichte auferlegt sind.

2.2.2 Beispiele für Klassifikatoren

Im Folgenden werden einige Klassifikatoren vorgestellt mit dem Ziel, einen Vergleich führen zu können und eine Begründung dafür zu liefern, warum RCE-Netzwerke für die Implementierung in Hardware besonders gut geeignet sind.

Bayes-Klassifikator

Der Bayes-Klassifikator fällt die Entscheidung der Klassenzuordnung auf Basis der Wahrscheinlichkeitsrechnung. Zuspruch erhält die Klasse ω_i, der $\vec{x}$ am ehesten zugeordnet werden kann. Für jede Klasse wird die a-posteriori-Wahrscheinlichkeit $P(\omega_i|\vec{x})$ ermittelt, die angibt, mit welcher Wahrscheinlichkeit ein Muster $\vec{x}$ in die Klasse ω_i fällt. $\vec{x}$ wird der Klasse $\omega_a \in \Omega$ zugeordnet, für die gilt:

$$\vec{x} \in \omega_a \Leftrightarrow P(\omega_a|\vec{x}) > P(\omega_i|\vec{x});\ i = 1, ..., k \wedge i \neq a \text{ mit } k \text{ Anzahl der Klassen} \quad (2.1)$$

In Abbildung 2.3 sind die a-posteriori-Wahrscheinlichkeiten zweier Klassen in einem ein dimensionalen Merkmalsraum eingezeichnet. Für ein gemessenes Muster tritt für jede Klasse eine bestimmte Wahrscheinlichkeit ein. Eine eindeutige Zuordnung ist nur dann nicht möglich, wenn das Muster auf die Trennungslinie fällt. Die Berechnung der

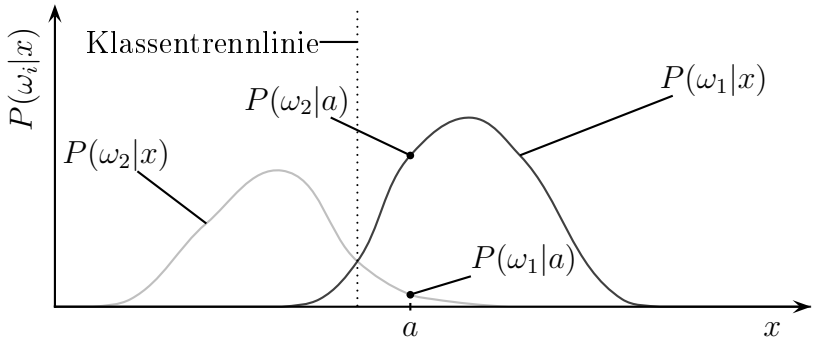


Abbildung 2.3: Bayes-Klassifikation

Wahrscheinlichkeit, dass ein Muster $\vec{x}$ aus einer Klasse ω_i stammt, erfordert folgende Vorkenntnisse:

- $P(\omega_i)$: Wahrscheinlichkeit für das Auftreten der Klasse ω_i.
- $P(\vec{x}|\omega_i)$: Klassenbedingte Wahrscheinlichkeit für das Auftreten von $\vec{x}$, wenn ω_i angenommen wird.
- $P(\vec{x})$: Wahrscheinlichkeit für das Auftreten von $\vec{x}$.

$P(\omega_i|\vec{x})$ errechnet sich dann wie folgt:

$$P(\omega_i|\vec{x}) = \frac{P(\omega_i) \cdot P(\vec{x}|\omega_i)}{P(\vec{x})} \quad (2.2)$$

Detailierte Informationen zum Thema Wahrscheinlichkeitsrechung und Statistik können in [Sac06] nachgelesen werden. Oft ist es schwierig, $P(\omega_i)$ und $P(\vec{x}|\omega_i)$ anzugeben, da sie eine genaue Kenntniss über den signalerzeugenden Prozess erfordern. Daher

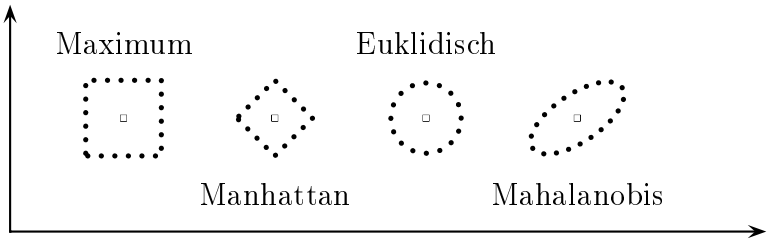


Abbildung 2.4: Gleiche Abstände verschiedener Abstandsmaße

ist der Bayes-Klassifikator eher theoretischer Natur und dient der Leistungseinschätzung anderer Klassifikatoren. Liegen sowohl $P(\omega_i)$ und $P(\vec{x}|\omega_i)$ vor, kann der Bayes-Klassifikator eingesetzt werden, und das Ergebnis kann durch keinen Klassifikator übertroffen werden.

Das Maximum-Likelihood-Verfahren ([DSH00]) ist aus dem Bayes-Klassifikator abgeleitet und kommt ohne Kenntnis der a-priori-Wahrscheinlichkeit $P(\omega_i)$ aus; ist aus diesem Grund auch nicht optimal, sondern lediglich eine Näherung.

Abstandsklassifikator

Sehr häufig kommen Abstandsklassifikatoren zum Einsatz. Hier wird die Klassenzugehörigkeit über den Abstand $\delta(\vec{x}, \bar{\omega}_i)$ der Muster zu den jeweiligen Klassenmittelpunkten errechnet, und es erhält die Klasse ω_a den Zuspruch, für die gilt:

$$\vec{x} \in \omega_a \Leftrightarrow \delta(\vec{x}, \bar{\omega}_a) < \delta(\vec{x}, \bar{\omega}_i);\ i = 1...M \wedge i \neq a \tag{2.3}$$

Welches Abstandsmaß verwendet wird, spielt zunächst keine Rolle. Möglich sind hier beispielsweise Euklidischer Abstand, Mahalanobis Abstand, Manhattan Distanz oder Maximum Norm, welche sich jedoch in Leistung und Rechenzeit stark unterscheiden. Abbildung 2.4 veranschaulicht Punkte, die zu ihrem Zentrum den jeweils gleichen Abstand haben. Sie unterscheiden sich nur bedingt durch die Art der Abstandsnorm. Der Mahalanobis-Abstand bietet zwar die besten Resultate, da neben dem Klassenzentrum auch die Streuung der Testdaten in die Berechnung einfließen. Die hohe Qualität geht jedoch zu Lasten der Rechenleistung, was in Hinsicht auf eine Implementierung in Hardware nicht außer Acht gelassen werden kann. Auch das euklidische Abstandsmaß ist aus dieser Sicht noch teuer, so dass vor allem solche, die ohne Multiplikationen auskommen, zu bevorzugen sind. In diesem Sinne ist z.B. die Manhatten-Distanz oder Maximumnorm als geeignet anzusehen. Die Manhattan- oder auch Cityblock-Distanz

zwischen zwei Mustern $\vec{a}$ und $\vec{b}$ der Dimension D errechnet sich als:

$$\delta(\vec{a},\vec{b}) = \sum_{i=0}^{D-1} abs(a_i - b_i) \tag{2.4}$$

und die Maximumsnorm als:

$$\delta(\vec{a},\vec{b}) = max\{abs(a_i - b_i)|i = 0,...,D-1\} \tag{2.5}$$

Für den späteren Entwurf des Klassifikators sind diese Distanzmaße wegen ihrer einfachen Berechnungsvorschrift besonders interessant. Eine Entscheidung zwischen beiden Möglichkeiten muss allerdings nicht getroffen werden, da sie äquivalent sind und durch eine affine Transformation des Merkmalsraums ineinander überführt werden können.

Box-Klassifikator

Die Abstandsklassifikatoren teilen den Merkmalsraum vollständig auf. Dieser Umstand liefert zwar ein hohes Generalisierungsvermögen, jedoch werden auch Muster in die existierenden Klassen eingeordnet, die sehr weit von den existierenden Klassenmittelpunkten entfernt sind. Praktisch sinnvoller wäre eine Erweiterung der Klassifikatoren um einen Schwellwert σ, der eine obere Schranke für die Entfernung zum Klassenzentrum festlegt.

$$\vec{x} \in \omega_a \Leftrightarrow \delta(\vec{x},\bar{\omega_a}) < \delta(\vec{x},\bar{\omega_i});\ i = 1,...,k \wedge i \neq a \wedge \delta(\vec{x},\bar{\omega_a}) < \sigma \tag{2.6}$$

Eine Realisierung dieser Erweiterung ist bei den Box-Klassifikatoren zu finden. Der Merkmalsraum wird nicht mehr vollständig aufgeteilt. Lediglich eine gewisse Region, beschrieben durch eine Fläche der Größe f, enthält alle Punkte im Merkmalsraum, die einer Klasse zugeordnet werden können.

Bei diesen Klassifikatoren gibt es demnach Regionen, die keiner Klasse zugeordnet werden können und es muss eine weitere Klasse $\omega_\emptyset$ eingeführt werden, die als sogenannte Rückweisungsklasse dient. Alle Muster, die nicht zuweisbar sind, fallen dann in $\omega_\emptyset$. Gegeben sei eine Menge von Testdaten $T = \{\vec{t}_1,...,\vec{t}_N\}$ der Dimension D, die bekanntermaßen der Klasse ω_a angehören. Daraus ergibt sich eine Hyperfläche mit dem Mittelpunkt

$$\bar{\omega}_{a,i} = \frac{1}{2}(min\{\vec{t}_{j,i}|j = 1,...,N\} + max\{\vec{t}_{j,i}|j = 1,...,N\}); i = 0..D-1, \tag{2.7}$$

für jede Dimension $i \in D$, und einem zugehörigen Radius

$$\mathring{\omega}_{a,i} = max\{\delta(\bar{\omega}_{a,i},\vec{t}_{j,i})|i = 0..D-1\}. \tag{2.8}$$

Eine Hyperfläche ist eine Fläche beliebiger Dimension. Oft wird auch von Hypercubes oder Hypersphären gesprochen. Abbildung 2.5 veranschaulicht dieses Verfahren für einen zwei-dimensionalen Merkmalsraum und einer Testdatenmenge für zwei Klassen. Die Hyperfläche konstruiert sich im Allgmeinen aus den Histogrammen in den einzelnen Dimensionen. Abbildung 2.5 zeigt jedoch auch, dass bedingt durch die Verteilung der Testdaten Überlappungen auftreten können. Muster, die in eine Schnittregion mindestens zweier Klassen fallen, müssen ebenfalls als unsicher zurückgewiesen und gesondert behandelt werden. Hier bietet sich wieder die Rückweisungsklasse $\omega_\emptyset$ oder eine separate Klasse für Mehrfachzuweisungen an.

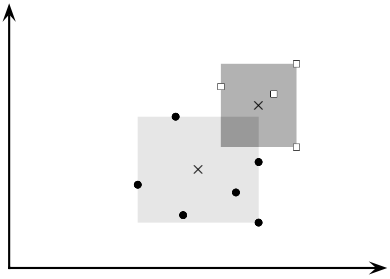


Abbildung 2.5: Merkmalsraumaufteilung des Box-Klassifikators

Restricted-Coulomb-Energy-Netzwerke (RCE)

Der Bayes-Klassifikator bietet den Vorteil, multimodale Klassenverteilungen realisieren zu können. Das bedeutet, dass eine Klasse nicht nur ein, sondern mehrere Klassenzentren haben kann. Abbildung 2.6 zeigt die Wahrscheinlichkeitsdichtefunktionen der a-posteriori-Wahrscheinlichkeiten für zwei Klassen ω_1 und ω_2. Da $P(\omega_1|x)$ mehrere Maxima und mehr als einen Schnittpunkt mit $P(\omega_2|x)$ hat, kommen separate Regionen zustande, die der Klasse ω_1 zugeordnet sind.

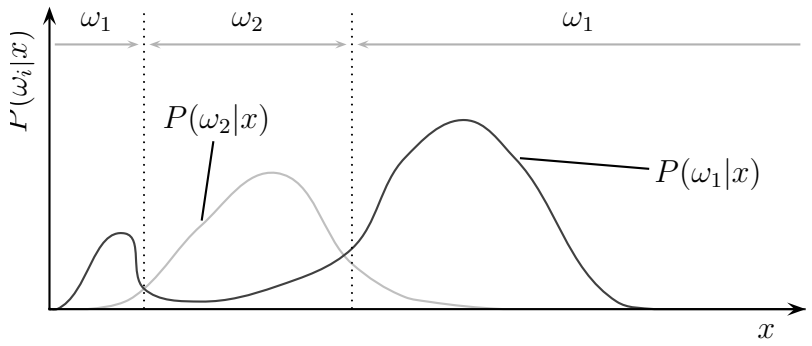


Abbildung 2.6: Multimodale Klassenverteilung beim Bayes-Klassifikator

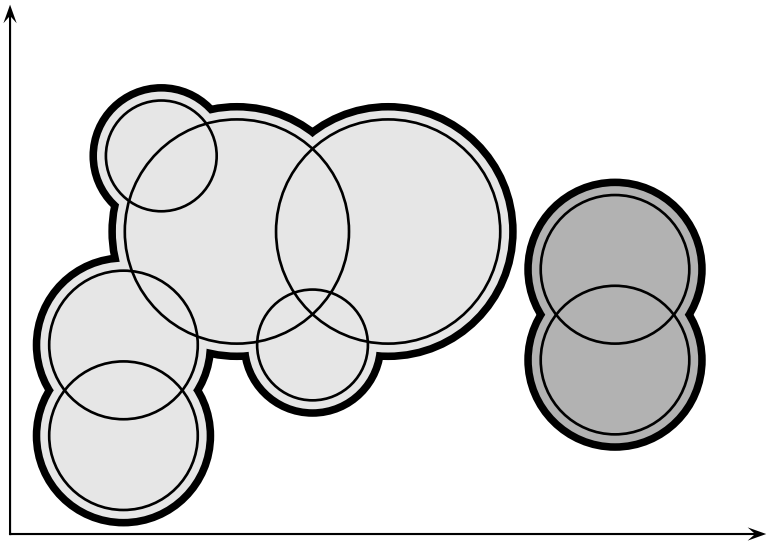


Abbildung 2.7: Merkmalsraumaufteilung bei RCE-Netzwerken

Die Box-Klassifikatoren zeigen dieses Verhalten nicht, da jede Klasse durch genau eine Hyperfläche beschrieben wird. Erweitert man die Box-Klassifikatoren um die Möglichkeit, für jede Klasse mehrere Hyperflächen anzulegen, ergibt sich auch hier die Fähigkeit, multimodale Klassenverteilungen erfassen zu können.

Die Restricted-Coulomb-Energy-Netzwerke (kurz: RCE-Netzwerke) erfüllen genau diese Bedingung. Für jede Klasse ω_i gibt es eine Menge $H_i = \{h_0, \ldots, h_{J-1}\}$ von J Hyperflächen, die ω_i im Merkmalsraum beschreiben. Jede Hyperfläche wird durch einen Referenzvektor $\vec{h}_l$ auf den Mittelpunkt und einen Radius $\mathring{h}_l$ beschrieben.

Fällt nun ein Muster $\vec{x}$ in den Einflussbereich einer der Hyperflächen H_i, d.h.

$$\exists h \in H_i : \delta(\vec{x}, \vec{h}) < \mathring{h} \Rightarrow \vec{x} \in \omega_i. \tag{2.9}$$

Abbildung 2.7 zeigt die Aufteilung eines zwei-dimensionalen Merkmalsraums in zwei Klassen ω_1 und ω_2. Die Verwendung der euklidischen Metrik ist auch hier wieder willkürlich. Die Verwendung mehrerer Hyperflächen liefert neben dem Vorteil der Multimodalität auch die Möglichkeit, unregelmäßige Hyperflächen im D-dimensionalen Raum beschreiben zu können. RCE-Netzwerke entscheiden, ob ein Muster in einer bestimmten Klasse enthalten ist oder nicht. Sie treffen keine Aussage über die Wahrscheinlichkeit. Im Falle überlappender Klassen gibt es daher Muster $\vec{x}$, für die gilt $\vec{x} \in \omega_i \wedge \vec{x} \in \omega_j; i \neq j$. Solche Muster müssen als unsicher zurückgewiesen und einer Klasse $\omega_\emptyset$ zugeordnet werden. Für den praktischen Einsatz ist es daher notwendig, den Klassifikator so zu konfigurieren, dass möglichst keine Überlappungen auftreten.

Nähere Details zur Konfiguration und Realisierung des Klassifikators werden in Kapitel

3 beschrieben. Ein Beispiel ist in Anhang D zu finden. An dieser Stelle sei vorweggegriffen, dass unter Verwendung der City-Block-Metrik ein leicht zu realisierender und leistungsstarker Klassifikator gegeben ist. Zum einen kann auf teure Operationen wie Multiplikationen verzichtet werden, und andererseits ist es möglich komplexe Klassenstrukturen zu beschreiben.

2.2.3 Fazit

Die RCE-Netzwerke unterscheiden sich, in Bezug auf das Klassifikationsergebnis, gegenüber den Bayes-Klassifikatoren vor allem dadurch, dass keine Aussage über die Wahrscheinlichkeit der Klassenzuordnung gemacht werden kann. Der Vorteil der Multimodalität als auch die Fähigkeit komplexe Klassenstrukturen beschreiben zu können, ist jedoch beiden gegeben. Diese Eigenschaften sind weder bei den Abstands- noch bei den Box-Klassifikatoren vorzufinden. Die hohe Qualität des Bayes-Klassifikators, die Wahrscheinlichkeit einer Fehlklassifikation zu minimieren, beruht auf der Verwendung von a-priori-Wissen, das in der Praxis nur selten gegeben ist. Es können auch nur Muster verarbeiten werden, zu denen Vorwissen bekannt ist. Daher fehlt jegliche Fähigkeit zur Generalisierung.

Die Abstandsklassifikatoren hingegen generalisieren in maximaler Weise. Jedes Muster $\vec{x}$ wird der Klasse $\omega_i \in \Omega$ zugeordnet, der es am nächsten ist. Dadurch wird aber der praktisch relevante Fall vernachlässigt, dass ein Muster mit zunehmender Entfernung zu einem Klassenzentrum auch an Ähnlichkeit verliert und ab einer gewissen Distanz zurückgewiesen werden sollte.

Die Box-Klassifikatoren und RCE-Netzwerke bilden hier einen Kompromiss und abstrahieren nur in einem Bereich, der durch die Testdaten belegt wurde. Besonders die RCE-Netzwerke (siehe Kapitel 3) bieten die Möglichkeit, durch eine geeignete Konfiguration das Abstraktionsniveau an die Sachlage anzupassen. Da es tatsächlich von der Problemstellung abhängig ist, in welchem Maße abstrahiert werden soll, ist diese Fähigkeit der RCE-Netzwerke besonders hervorzuheben. Sie bieten jedoch nicht die Möglichkeit, maximal zu abstrahieren. Für dieses Anwendungsfeld sind tatsächlich die Abstandsklassifikatoren zu bevorzugen.

2.3 Field Programmable Gate Arrays (FPGAs)

Bei den *Field-Programmable-Gate-Arrays* (kurz FPGAs) handelt es sich um eine sehr junge Bausteinfamilie, die sich vor allem durch ihre Konfigurierbarkeit auszeichnet. 1985 brachte die Firma Xilinx den ersten kommerziellen FPGA auf den Markt, der mit knapp 1000 Gatteräquivalenten (s.u.) noch sehr bescheiden bestückt war. Vor dieser Zeit war es notwendig, Standardbausteine auf einer Platine zu verdrahten oder eine spezielle integrierte Schaltung herzustellen [Wan98].

Abbildung 2.8: Cyclone II FPGA - Modell EP2C20F484C7

Um die Leistung von FPGAs anzugeben, erwies sich das *Gatteräquivalent* als geeignet. Es gibt an, wieviele 2NAND-Gatter auf einem Baustein untergebracht werden können. Dieses Maß stammt aus der Zeit der *Gate Arrays*. Die Logikelemente (LE) der FPGAs bestehen aber nicht wie Gate Arrays aus einzelnen Gattern, sondern aus einer mehr oder weniger komplexen Logik, wodurch dieses *Gatteräquivalent* einen hohen Spielraum bietet.

Heute sind FPGAs erhältlich, die eine Million Gatteräquivalente aufweisen.

Ihren hohen Beliebtheitsgrad erlangten FPGAs vor allem weil sie keine anwendungsspezifischen Bausteine waren, sondern durch reines Konfigurieren an eine gegebene Aufgabenstellung angepasst werden konnten. Es wurde damit möglich, ein und dieselbe Hardware für unterschiedliche Aufgaben einzusetzen oder Logikfehler bei Bedarf zu korrigieren.

Bei FPGAs spricht man eher von Konfigurieren statt von Programmieren, da keine neuen Strukturen auf den Chips erzeugt werden. Es wird nur festgelegt, wie die vorhandenen Elemente verschaltet werden. Die Marktführer Altera und Xilinx bieten Entwicklungstools an, mit denen komfortabel der Hardwareentwurf für FPGAs in

Merkmal	**Cyclone II EP2C20F484C7**	**Stratix III EP3SL150F1152C2NES**
Logikelemente	20.000	142.000
Preis	52$	3.277 $
on-Chip-Speicher	240 kb	5499 kb
on-Chip-Multiplizierer	26	384
User I/O - Pins	315	480

Tabelle 2.1: zwei Altera-FPGAs im Vergleich

VHDL oder Verilog möglich ist. Neben Quelltext-Editoren bieten sie auch grafische Tools, mit denen einzelne Komponenten platziert und verbunden werden können.

2.3.1 Allgemeiner Aufbau

Im wesentlichen bestehen FPGAs aus Logikelementen, die in einer bestimmten Struktur (hierarchisch, matrixform usw.) auf dem Chip angeordnet sind. Sie sind durch eine konfigurierbare Verbindungsstruktur miteinander verdrahtet. Zur Kommunikation mit der Peripherie stehen I/O-Elemente zur Verfügung, die je nach Bedarf als Input oder Output genutzt werden können [KB06]. Zu den führenden Herrstellern von FPGAs zählen heute die Firmen Xilinx und Altera. Für diese Arbeit wird exemplarisch auf einem Cyclone-II (EP2C20) der Firma Altera gearbeitet, der als *Low-Cost*-Produkt nicht in den High-End-Bereich fällt. Altera vertreibt derzeit drei FPGA-Produktfamilien: Cyclone, Stratix und Arria. Die Preisspanne sowie die Leistungsunterschiede sind dabei erheblich. Cycone-II Devices gibt es mit 4600 - 68000 LEs und variieren entsprechend zwischen 12 und 300 $. Je nach Ausführung verfügen die Chips über mehr oder weniger viele eingebettete Multiplizierer und *on-chip-Memory*-Blöcke (M4K-Blöcke). Die High-End Produkte von Altera - die Stratix III Produktfamilie - verfügen hingegen über bis zu 340.000 LEs. Dafür sind dann auch Preise von bis zu 4000 $ möglich. Tabelle 2.1 stellt den EP2C20 einem High-End-Produkt - dem EP3SL150 - gegenüber.

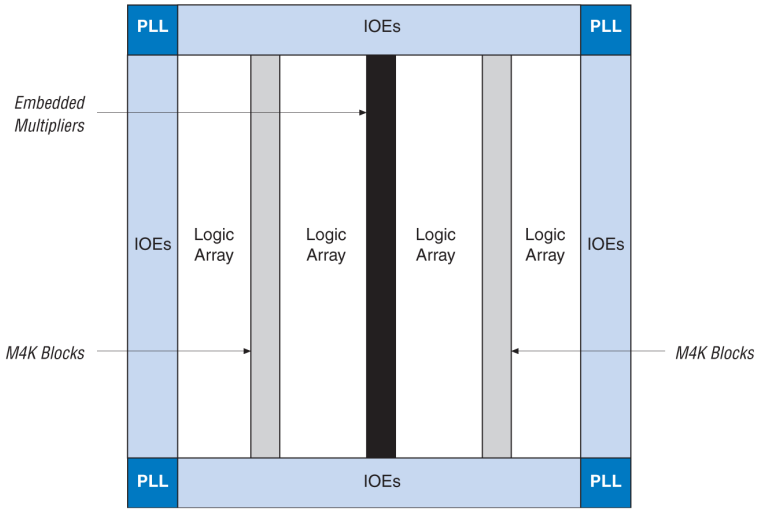


Abbildung 2.9: Cyclone II EP2C20 - Blockdiagramm (Quelle: [Alt04])

2.3.2 Architektur am Beispiel der Cyclone II Familie von Altera

Die Cyclone-II Familie basiert auf einer 90-nm SRAM-Technologie mit bis zu 68.000 LEs. Durch die Integration von 18×18 Multiplizierern und M4K-Blöcke (siehe Abb. 2.9) werden DSP-Applikationen gut unterstützt. Zur Kommunikation mit SRAM und DRAM-Speichern stehen schnelle Schnittstellen zur Verfügung, die einen Datendurchsatz von 333 Mbps erlauben.

Die Cyclone-II Bausteine bestehen aus einer Matrix-Struktur. Je 16 LEs werden zu sogenannten *logic array blocks* (LABs) zusammengefasst. Ein LAB kann beispielsweise direkt als 32-Bit-Volladdierer genutzt werden. Zur Kommunikation zwischen den LABs, dem on-Chip-Speicher (M4K-Blöcke), den Multiplizierern und den I/O-Elementen (IOEs) dient ein Verbindungsnetz. Sie verfügen über ein globales Clock-Netzwerk, mit dem bis zu 16 verschiedene Frequenzen zeitgleich verwendet werden können.

Logikelemente

Anzahl und Leistungsfähigkeit der LEs variiert je nach Hersteller und Fabrikat. Allgemein sind sie so konzipiert, dass ihnen im Rahmen der verfügbaren Ressourcen jedes digitalelektronische Verhalten zugewiesen werden kann. Heute existieren LEs auf Basis von *Look-Up-Tables* (kurz LUTs) oder Multiplexern.

Abbildung 2.10 zeigt ein Logikelement der Cyclone II Familie von Altera. Im Kern besteht das LE aus einer LUT und einem Register. Die LUT verfügt über vier Eingänge und kann somit vierstellige Funktionen realisieren.

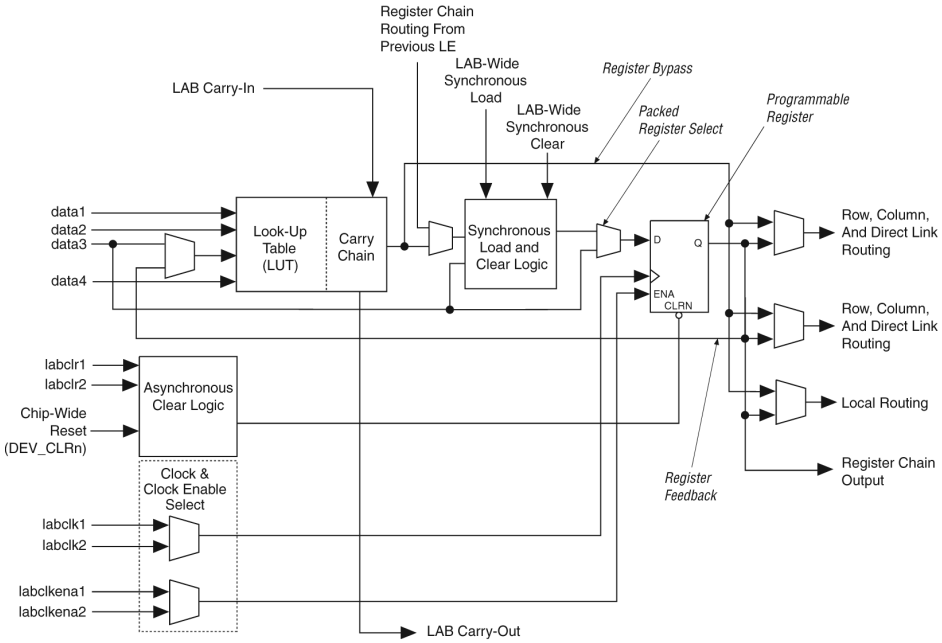


Abbildung 2.10: Logik-Element der Cyclone-II Familie von Altera

3

Der RCE-Netzwerk-Klassifikator

Kapitel 2.2.2 konnte bereits zeigen, dass die RCE-Netzwerke gegenüber anderen Verfahren einige Vorteile bieten. Im folgenden soll die Arbeitsweise des Klassifikators genauer betrachtet werden.

Die *Restricted-Coulomb-Energy* (RCE)-Netzwerke stellen im Kern das *Nestor-Lerning-System* dar [Nes95]. Zur Anwendung kommt dieses Verfahren in dem *Neuro-Chip* NI1000, welcher zur Klassifizierung hochdimensionaler Muster dient. Die Bezeichnung *Coulomb-Energy* stammt aus der Elektrophysik und beschreibt die radiale Abhängigkeit geladener Teilchen, deren Einfluss auf eine gewisse Entfernung beschränkt (*restricted*) ist [Hal99].

Während der Lernphase wird aus Trainingsdaten eine Menge von Hyperflächen $H_i = \{h_0, \ldots, h_{j-1}\}$ generiert, welche die Klassenstruktur von ω_i approximiert. Jede Hyperfläche h_l ist ein Tupel $h_l = (\vec{h}_l, \mathring{h}_l)$, wobei $\vec{h}_l$ ein Referenzvektor auf den Mittelpunkt der Hyperfläche und $\mathring{h}_l$ deren Radius ist. Jede Klasse kann durch eine beliebige Anzahl von Hyperflächen beschrieben werden.

3.1 RCE-Lernverfahren

Der Konfiguration des Klassifikators liegt ein Algorithmus zu Grunde, der aus den Trainingsdaten schrittweise Hyperflächen generiert, deren Eignung zur Beschreibung der realen Klassenstrukturen von der Repräsentativität der Trainingsdaten abhängt.

Der Algorithmus arbeitet im wesentlichen wie folgt [Nes95]:

Vor dem Lernvorgang müssen durch den Anwender die radialen Grenzen r_{min} und r_{max} (siehe Abschnitt 3.3) festgelegt werden. Während des Lernvorgangs wird für jeden Trainingsdatenvektor die Distanz δ zu bereits existierenden Referenzvektoren ermittelt. δ ist die Summe der Differenzen der Komponenten eines Trainingsvektors und der korrespondierenden Komponenten eines Referenzvektors. Diese Manhattan-Metrik berechnet sich als:

$$\delta(\vec{a}, \vec{b}) = \sum_{i=0}^{D-1} |a_i - b_i| \tag{3.1}$$

Die Distanzen werden mit den zugehörigen Radien verglichen, um so zu bestimmen, ob und in welchen Einflussbereich bereits bestimmter Hyperflächen der Trainingsvektor fällt. Falls er durch keine Hyperfläche erfasst werden kann, wird er selbst als Repräsentant seiner eigenen Klasse, mit Radius r_{max}, gespeichert. In dem Fall, dass er in den Einflussbereich einer fremdklassigen Hyperfläche fällt, wird er in die Menge H aufgenommen, und die Radien beider werden auf deren Distanz gesetzt. Obwohl sie sich nicht mehr gegenseitig erfassen, kommt es zu einer Überlappung zwischen beiden Hyperflächen. Der Algorithmus nimmt keine neue Hyperfläche auf, wenn der Trainingsvektor nur in den Einflussbereich von Flächen fällt, die der eigenen Klasse angehören.

Der Algorithmus differenziert also die folgenden vier Fälle, nach denen ein Trainingsvektor verarbeitet wird [Hal99]:

1. Der Trainingsvektor fällt in keinen bereits existierenden Einflussbereich: Eine neue Hyperfläche mit r_{max} als Radius wird angelegt.
2. Der Trainingsvektor fällt in den Einflussbereich existierender Hyperflächen der eigenen Klasse: Es wird keine Veränderung an den Hyperflächen vorgenommen.
3. Der Trainingsvektor fällt in den Einflussbereich einer fremdklassigen Hyperfläche: Der Trainingsvektor wird als Vertreter seiner eigenen Klasse aufgenommen, und die Radien beider Hyperflächen werden auf deren Distanz gesetzt.
4. Der Trainingsvektor fällt in den Einflussbereich eigener und fremder Hyperflächen: Der Radius der fremden Hyperflächen wird auf die Distanz verkleinert, aber es wird keine neue Hyperfläche angelegt.

Wenn Radien verkleinert werden, dann gilt immer r_{min} als untere Schranke, d.h. jeder Radius ist $\geq r_{min}$. Nachdem alle Trainingsvektoren einsortiert wurden, wird erneut begonnen, dieselbe Menge von Trainingsvektoren einzuordnen. Auf diese Weise wiederholt sich der Algorithmus so lange, bis keine Änderungen an den Hyperflächen und deren Radien mehr stattfindet. Durch das wiederholte Ausführen des Algorithmus können noch bestehende Überlappungen minimiert werden [Hal99].

3.2 RCE-Klassifikation

Während der Klassifikation verbirgt sich hinter jeder Hyperflächen eine Aktivierungsfunktion

$$\lambda(\vec{x}, \vec{h}_l, \mathring{h}_l, r_{min}) = \begin{cases} 1, & \text{wenn } (\delta(\vec{x}, \vec{h}_l) < \mathring{h}_l) \wedge (\mathring{h}_l > r_{min}); \\ 0, & \text{sonst} \end{cases} \tag{3.2}$$

die angibt, ob $\vec{x}$ in den Einflussbereich von h_l fällt. Im Folgenden wird sie oft nur als $\lambda(h_l)$ bezeichnet, da sie während einer Klassifikation nur von der Hyperfläche abhängt. Die Größe r_{min} gibt die Mindestgröße einer Hyperfläche an und resultiert aus den Anforderungen an des Generalisierungsverhalten (siehe Abschnitt 3.3). Die Ausgänge der einzelnen Hyperflächen werden disjunktiv verknüpft und $\vec{x}$ wird ω_i zugeordnet, wenn es in den Einflussbereich einer der Hyperflächen aus H_i fällt:

$$\vec{x} \in \omega_i \Leftrightarrow \bigvee_{h_l \in H_i} \lambda(h_l) = 1 \tag{3.3}$$

Muster, die in keine der Klassen Ω fallen, werden einer Rückweisungsklasse $\omega_\emptyset$ zugeordnet.

Gleichung 3.2 bringt auch zum Ausdruck, dass eine Hyperfläche mit einem Radius $\leq r_{min}$ seine Bedeutung verloren hat und für eine Zuordnung nicht länger in Frage kommt.

3.3 Generalisierungsverhalten

Über die beiden Parameter r_{min} und r_{max} wird das Lernverhalten und die entstehende Konfiguration maßgeblich bestimmt. Diese Werte sind für jedes Klassifikationsproblem neu zu bestimmen und von den signalerzeugenden Prozessen abhängig.

Mit Hilfe von r_{max} kann festgelegt werden, in welchem Maße generalisiert werden soll, d.h. wie groß die Distanz zwischen einem Referenzvektor und einem Muster maximal sein kann, damit das Muster noch als Exemplar dieser Klasse erkannt wird. Wird r_{max} sehr groß gewählt, entstehen zunächst sehr große Hyperflächen, die erst dann verkleinert werden, wenn sie durch Trainingsdaten anderer Klassen gestört werden. Dadurch kann bei sehr groß gewählten Werten für r_{max} erreicht werden, dass das System so weit wie möglich generalisiert. Existieren mindestens zwei Klassen und wird $r_{max} = \infty$ gewählt, so verringert sich der Radius einer Hyperfläche auf den kleinsten Abstand zu einem Trainingsvektor einer fremden Klasse. Kleine Werte von r_{max} bewirken, dass die

Hyperflächen sehr klein sind und unter Umständen viele Hyperflächen nötig sind, um das Klassengebiet abzudecken.

Hingegen legt r_{min} fest, in welcher Umgebung eines Referenzvektors keine fremden Trainingsdaten auftreten dürfen. Durch r_{min} wird damit eine Ähnlichkeits-Umgebung vorgeschrieben, die Grundbedingung ist. Innerhalb dieser Umgebung werden keine fremden Trainingsvektoren akzeptiert. Wird r_{min} sehr klein gewählt, sind prinzipiell alle Klassifikationsprobleme separierbar. Es entsteht aber der Nachteil, dass der Merkmalsraum, wenn nötig, durch punktförmige Hyperflächen beschrieben wird. Eine zu große Wahl von r_{min} legt einen entsprechend großen Mindestradius für die Hyperflächen fest. Eine Klassifikationsaufgabe kann sehr schnell zu einem nicht separierbaren Problem werden, sobald r_{rmin} den kleinsten Abstand zwischen zwei Trainingsvektoren unterschiedlicher Herkunft übersteigt. Wird $r_{min} = \infty$ maximal gewählt, entstehen gar keine Hyperflächen.

4

Realisierungsvarianten

Für die Berechnung der Manhattan-Distanz zweier Vektoren $\vec{a}$ und $\vec{b}$ sind die paarweisen Differenzen ihrer Komponenten zu addieren. Die Dimension des Merkmalsraums ist keine feststehende Größe und kann bei jeder Klassifikationsaufgabe anders ausfallen. Die Anzahl der Hyperflächen M ist ebenso unbekannt. Für die Entscheidung, ob und in welche Klasse das aktuelle Muster fällt, sind die Aktivierungsfunktionen aller Hyperflächen miteinander zu vergleichen.

Beide Aufgaben haben die Eigenschaft, eine variable Anzahl von Werten auf eine bestimmte Weise zu verknüpfen. Grundsätzlich gibt es verschiedene Varianten, wie diese Problemstellungen gelöst werden können. Am Beispiel der Manhattan-Distanz werden im Folgenden verschiedene Konzepte vorgestellt, die für den Entwurf in Frage kommen, sich in Zeit- und Platzverhalten aber unterscheiden.

4.1 Serielle Distanzberechnung

Die D-dimensionalen Vektoren $\vec{a}$ und $\vec{b}$ werden zunächst serialisiert und die einzelnen Komponenten a_i und b_i werden taktweise verarbeitet. Dazu wird der Absolutbetrag der Differenz beider Komponenten $|a_i - b_i|$ auf den Inhalt eines Registers addiert. Das Resultat wird anschließend in dem selben Register abgelegt. Nach d Schritten wurden alle Komponenten verarbeitet, und die Distanz zwischen $\vec{a}$ und $\vec{b}$ liegt am Ausgang an. Bevor eine neue Distanz berechnet werden kann, muss das Register zurückgesetzt werden.

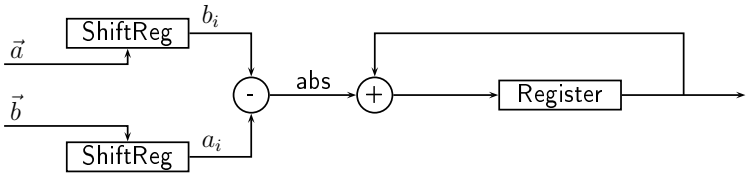


Abbildung 4.1: serielle Summenbildung

Der große Vorteil dieser Variante ist der konstante Platzbedarf, da lediglich ein einziges Register für die Speicherung der Zwischensumme notwendig ist. Die Anzahl der Dimensionen hat keinen Einfluss auf den Platzbedarf. Für die Berechnung der Distanz werden d Takte benötigt. Während dieser Zeit können keine weiteren Vektoren verarbeitet werden. Diese Art der Distanzberechnung ist dann sinnvoll, wenn der signalerzeugende Prozess höchstens alle d Takte ein neues Muster liefert, andernfalls ist diese Variante ungeeignet. Die resultierende Verzögerung d ist unter dem Gesichtspunkt der Echtzeitsignalverarbeitung weniger das Problem, sondern vielmehr der Umstand, dass für die Dauer einer Berechnung keine weiteren Daten verarbeitet werden können.

4.2 Geteilt-serielle Distanzberechnung

Echtzeitsignalverarbeitung erfordert, dass eingehende Daten verarbeitet werden, ohne dass ein nicht zubewältigender Datenstau entsteht. Dieses Ergebnis kann nur erreicht werden, wenn mit der Geschwindigkeit Daten verarbeitet werden, mit der der Eingang Daten liefert. Die serielle Distanzberechnung (kurz: SD) benötigt hingegen d Takte. Der Zeitbedarf kann halbiert werden, indem zwei SD-Module zeitgleich je eine Hälfte der Dimensionen verarbeiten und die Teilergebnisse hinterher addiert werden (Abbildung 4.2). Dadurch lässt sich der Datendurchsatz verdoppeln. Es verdoppelt sich allerdings auch der Platzbedarf.

Allgemein gilt, dass Platz- und Zeitverhalten stets voneinander abhängen und eine Verbesserung der einen Größe zu einer Verschlechterung der anderen führt.

Die einzelnen SD-Module berechnen in $\frac{d}{2}$-Takten die Teilergebnisse, die zunächst in einem Register gespeichert werden. Dadurch stehen sie für die nächsten $\frac{d}{2}$-Takte der nachfolgenden Addition zur Verfügung. Das Resultat wird abschließend wieder in einem Register gespeichert.

An der Latenz der Berechnung hat sich nichts geändert. Nach $\frac{d}{2}$-Takten liegt das Ergebnis der SD-Module in den beiden Registern REG_a und nach weiteren $\frac{d}{2}$-Takten im Register REG_b. Im selben Schritt speichern die Register REG_a bereits die Teilergebnisse der nächsten Berechnung. Wichtig ist hier, dass die Register nur alle $\frac{d}{2}$-Takte laden,

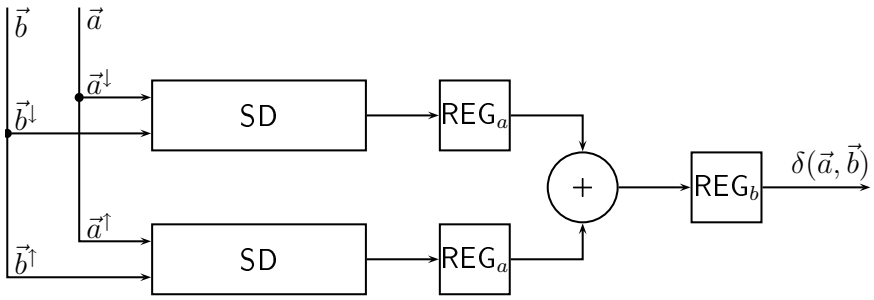


Abbildung 4.2: geteilt-serielle Summenbildung

da die SD-Module mit dieser Geschwindigkeit Daten liefern. Ein Datum hält sich nur noch für die Dauer von $\frac{d}{2}$-Takte in einer Komponente auf, läuft dann in die nächste und macht somit Platz für das nächste Datum. Dieses Prinzip ist auch als Pipelining bekannt, dass in Abschnitt 5.1 genauer vorgestellt wird.

Werden die geteilten Vektoren erneut halbiert, halbiert sich wiederum die Verarbeitungszeit der SD-Module und die Datenrate verdoppelt sich.

4.3 Distanzberechnung durch eine Baumstruktur

Dieses Prinzip kann maximiert werden, indem so lange halbiert wird, bis jeweils nur noch eine Dimension übrig bleibt. Die Berechnung der Distanz zwischen zwei Skalaren kann dann in einem Takt ausgeführt werden. Wenn zwei D-dimensionale Vektoren verarbeitet werden, so liegen nach vollständiger Teilung und Berechnung der Teildistanzen d Werte vor, die aufsummiert werden müssen. In analoger Weise, wie bei der geteilt-seriellen Berechnung werden die einzelnen Werte durch Addition wieder zusammengeführt, bis nur noch ein Wert übrig bleibt. Dieser paarweisen Zusammenführung von Teilergebnissen entspricht eine Baumstruktur der Tiefe $\log_2 D$. Auf diese Weise entsteht ein System, das nach einer gewissen Latenz in jedem Takt ein Ergebnis liefert. Nachteil dieser Variante ist ein linear wachsender Platzbedarf $\mathcal{O}(D)$.

Durch dieses Verfahren kann der Datendurchsatz maximiert werden. Die Auswertung der Aktivierungsfunktionsergebnisse und die Berechnung der Distanzen im Sinne des Pipelinings werden daher durch eine Baumstruktur umgesetzt. Die Werte der Blätter können dann in einem Takt berechnet werden.

5 Systemspezifikation

Das folgende Kapitel widmet sich der detaillierten Beschreibung des Hardwarelayouts. Dafür wird zunächst die Schnittstelle beschrieben und darauf aufbauend alle weiteren Komponenten. Auf diese Weise soll gezeigt werden, wie die aus den Betrachtungen des Algorithmus (Kapitel 3) folgenden Teilaufgaben in Hardware umgesetzt werden. Ziel ist es, den Signalfluss von der Eingabe bis hin zur Ausgabe deutlich zu machen, um ein umfassendes Verständnis für die Schaltung zu ermöglichen.

5.1 Das I/O - Interface

Abbildung 5.1 zeigt den Klassifikator und die aus den Anforderungen folgende Schnittstelle. Das `clock`-Signal ist ein beliebiger Takt der lediglich der Pipelinesteuerung innerhalb des Klassifikators dient. Die Schaltung ist im wesentlichen eine Aneinanderreiung von Logikbausteinen, die über Register den Datentransport realisieren.

Pipelining

Das sogenannte *Pipelining* ist eine Methode der Leistungssteigerung und orientiert sich an der aus der Industrie bekannten Fließbandarbeit. Der gesamte Prozess wird in Einzelschritte zerlegt und für jeden dieser Teilschritte existiert eine separate Einheit. Die Einheiten sind dabei so konzipiert, dass sie die am Eingang anliegenden Daten verarbeiten. Ein Register am Ausgang einer Einheit übernimmt bei steigender Taktflanke

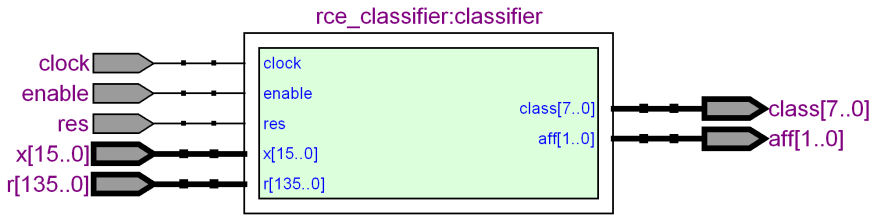


Abbildung 5.1: Die Schnittstelle des Klassifikators

das Resultat und stellt es für die Dauer eines Taktes der nächsten Einheit bereit (siehe Abbildung 5.2). Ein Muster wird dann verarbeitet, indem es nacheinander alle Einheiten durchläuft. Die Einheiten arbeiten völlig unabhängig voneinander, so dass eine Einheit nach der Bearbeitung eines Musters direkt das nächste verarbeiten kann. Dieses Prinzip sorgt dafür, dass nach einer gewissen *Latenz* (Verzögerungszeit) in jedem Takt ein Ergebnis geliefert wird. Die Latenz entspricht der Anzahl der Pipelinestationen und gibt an, wie lange es dauert, bis am Eingang angelegte Daten am Ausgang sichtbar werden.

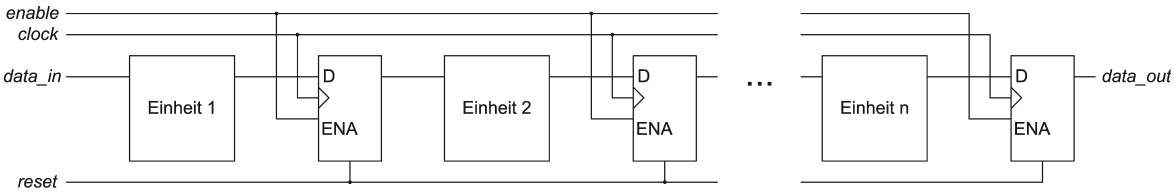


Abbildung 5.2: Pipeline

Eingänge

Neben dem `clock`-Signal werden die Register auch durch die `reset`- und `enable`-Signale gesteuert. Das `reset` wird benötigt, um das System in einen initialen Zustand zu versetzen. Dazu werden alle Register zurückgesetzt.

Der Klassifikator selbst ist ein Modul, das in einer Signalverarbeitungskette platziert wird und mit anderen Modulen verbunden werden kann. Es ist daher denkbar, dass ein Modul am Ausgang blockiert ist und der Klassifikator angehalten werden muss, bis er weitere Daten am Ausgang bereitstellt. Um diese Unterbrechung umzusetzen, existiert das `enable`-Signal, durch das gesteuert wird, ob ein Register ein neues Datum aufnehmen soll oder nicht. Die Pipeline kommt dann zum Stillstand, wenn das `enable`-Signal nicht gesetzt ist.

Generische Parameter

Der Klassifikator verfügt über drei generische Parameter (`D`, `M` und `N`), die vor der Synthese durch den Anwender festgelegt werden müssen. Sie werden an dieser Stelle vorgestellt, da deren Bedeutung für die Beschreibung der Ein- und Ausgänge klar sein muss. Ein sogenannter *Generic* macht eine Komponente allgemeiner und ermöglicht eine den Bedürfnissen angepasste Synthetisierung. Generics werden häufig eingesetzt, um die Bitbreite von Signalen festzulegen oder die rekursive bzw. iterative Instanziierung von Bausteinen zu steuern.

Der in Abschnitt 1.1 vorgestellte Neurochip NI1000 hat einige Beschränkungen, die durch diese VHDL-Implementierung unter Verwendung von Generics aufgehoben werden sollen [Nes95]:

1. Merkmale werden mit 5 Bit aufgelöst
2. es kann zwischen 64 verschiedenen Klassen separiert werden
3. der Merkmalsraum kann bis zu 256 Dimensionen haben
4. maximal können 8192 Referenzvektoren gespeichert werden, wenn die Dimension entsprechend niedrig ist

Die Beschränkung auf 5 Bit bei der Auflösung ist offensichlich die größte Schwachstelle. Der Klassifikator soll aber mit einer individuell gewünschten Bitbreite arbeiten. In der Entity des Klassifikators gibt es daher einen generischen Parameter `N`, über den die gewünschte Bitbreite für die Auflösung der Merkmale angegeben wird. Alle untergeordneten Komponenten werden mit diesem Parameter instanziiert und verfügen über davon abhängige Ein- und Ausgänge.

Weiterhin muss für jedes Klassifizierungsproblem die Dimension `D` des Merkmalsraums festgelegt werden. Ein `D`-dimensionales Muster $\vec{x}$ besteht aus `D` Komponenten:

$$\begin{aligned} \underline{x} &= [\underline{x_{D-1}} \circ \underline{x_{D-2}} \circ \ldots \circ \underline{x_1} \circ \underline{x_0}] \quad mit \\ &\quad \Theta(\underline{x_i}) = \mathtt{N} \quad \text{und} \quad \Theta(\underline{x}) = \mathtt{N} \cdot \mathtt{D} \end{aligned} \tag{5.1}$$

Es gilt $\underline{a}$ die binäre Repräsentation von a, $[a_1 \circ a_2]$ die Konkatenation von a_1 und a_2 zu einem neuen Wort und $\Theta(b)$ die Bitbreite von b.

Diese Repräsentation von Vektoren gilt für alle auftretenden Vektoren. Der Platz- und Zeitbedarf der Schaltung ist weiterhin von der Anzahl der Hyperflächen abhängig, die über den Parameter `M` angegeben wird. Dieser bewirkt unter anderem, dass für jeden Referenzvektor in der **DCU** (Distance Calculation Unit) eine *Activation Unit* (siehe Abschnitt 5.4.1) instanziiert wird.

Es sind also die folgenden drei Parameter, die für jede Klassifizierungsaufgabe angepasst werden müssen:

- `N` : Bitbreite der Merkmalsauflösung
- `D` : Dimension des Merkmalsraums
- `M` : Anzahl der Referenzvektoren

Die Breite des Datensignals `x` ergibt sich direkt aus den beiden Parametern `N` und `D`. Ist ein `D`-dimensionaler Merkmalsraum gegeben und soll jedes Merkmal mit `N` Bit aufgelöst werden so ist der Eingang x ein `N·D` breites Signal.

Die während der Trainingsphase entstehenden Hyperflächen sollen den Klassifikator auf die gewünschte Klassifizierungsaufgabe vorbereiten. Dazu müssen die einzelnen Hyperflächen in irgendeiner Weise durch die Schaltung eine Entsprechung finden. Hier sind prinzipiell zwei Varianten der Konfiguration denkbar:

Hard-Konfiguration Die Aktivierungsfunktionen hinter den Hyperflächen werden derart implementiert, dass die einzelnen Parameter ($\vec{h}_i, \mathring{h}_i$ und $\tilde{h}_i$) fester Bestandteil der Schaltung sind und nicht verändert werden können. Dadurch wird auf eine Konfiguration während der Klassifikation verzichtet, da das Hardwarelayout entsprechend der Trainingsdaten generiert wurde. Es kann sogar zu einer Verkleinerung des Platzbedarfs kommen, da die Kenntnis über Konstanten eine erhöhte Optimierung des Systems durch das Synthese-Tool erlaubt.

Soft-Konfiguration Die Aktivierungsfunktionen hinter den Hyperflächen werden durch sogenannte *Activation Units* (kurz AU) realisiert, die über entsprechende Eingänge zur Laufzeit konfiguriert werden. Dadurch kann eine AU bei veränderten Merkmalsraumaufteilung für andere Hyperflächen genutzt werden. Dann ist allerdings ein Konfigurationseingang notwendig, welche die einzelnen AUs mit Parametern versorgt, die abhängig von der gewählten Bitbreite, der Dimension des Merkmalsraum und der Anzahl der AUs sehr breit werden kann.

Die Fähigkeit zur Neukonfiguration überwiegt gegenüber der Möglichkeit, auf eine Datenleitung für die Konfiguration verzichten zu können. Praktisch relevant ist der Fall, eine Schaltung entsprechend einer absehbaren Komplexität zu erstellen und während der Anwendung nicht wieder neu zu synthetisieren. Ergibt sich auf Grund neuer Daten eine veränderte Merkmalsraumaufteilung, so ist es von Vorteil, diese Neukonfiguration dem Klassifikator bekannt zu machen, ohne die Schaltung resynthetisieren zu müssen.

Die Breite der Konfigurationsleitung $\mathtt{r}$ ergibt sich aus allen Größen, die sich während einer neuen Durchführung des Trainingsalgorithmus ändern können. Hierzu gehören

1. r_{min} und
2. Die Menge H der Hyperflächen.

Für die Konfiguration und den Eingang für das Muster wird je nur ein Signal verwendet. Das liegt vor allem daran, dass die Breite der Signale durch Generics angepasst werden kann, deren Anzahl jedoch nicht. Selbst interne Komponenten geben ihr Resultat immer auf einem entsprechend breiten Signal weiter und konkatenieren wenn nötig einzelne Datenworte zu einem.

Ein Datenwort $\mathtt{r}'$ der Konfigurationsleitung $\mathtt{r}$ hat zunächst folgenden Aufbau:

$$\mathtt{r}' = [\underline{r_{min}} \circ \underline{H}] \tag{5.2}$$

Durch eine neue Lernsituation kann sich die Anzahl $\#(H)$ der Hyperflächen ändern, wodurch auch die Bitbreite $\Theta(\mathtt{r}')$ ihren Wert ändert. Dieser Fall darf jedoch nicht eintreten, da nach der Synthese eine Datenleitung fester Breite vorliegt.

Wenn der Fall eintritt, dass $\#(H) < \mathtt{M}$ gilt, so muss H um eine $(\mathtt{M} - \#(H))$-elementige Menge $H_{neutral}$ neutraler Hyperflächen erweitert werden. Aus Gleichung 3.2 geht hervor, dass eine Hyperfläche nicht aktiviert wird, wenn deren Radius kleiner r_{min} ist. Daher kann eine Hyperfläche mit Radius 0 immer als neutrales Element verwendet werden.

Die Wahl von $\mathtt{M}$ sollte stets so erfolgen, dass eine veränderte Merkmalsraumaufteilung mit einer größeren Anzahl von Hyperflächen möglich ist. Generell gilt $\#(H) \leq M$ und $\#(H) + \#(H_{neutral}) = \mathtt{M}$.

Jede Hyperfläche $h_l \in H$ ist ein Tupel $h_l = (\vec{h}_l, \mathring{h}_l, \tilde{h}_l)$, wobei $\tilde{h}_l$ (mit $\omega_{\tilde{h}_l} \in \Omega$) die Bezeichnung der Klasse ist, der h_l zugeordnet ist. Die binäre Repräsentation von H ist die Folge

$$\begin{aligned} \underline{H} &= [\underline{h_{k-1}} \circ \underline{h_{k-2}} \circ \ldots \circ \underline{h_0}] \text{ mit} \\ \underline{h_l} &= [\underline{\tilde{h}_l} \circ \underline{\mathring{h}_l} \circ \underline{\vec{h}_l}]. \end{aligned} \tag{5.3}$$

Die Komponenten der Vektoren, die Radien sowie die Klassenbezeichnungen werden mit $\mathtt{N}$ Bit aufgelöst.

Insgesamt wird die Breite $\Theta(r)$ der Konfigurationsleitung durch folgende Bestandteile bestimmt:

- `N` Bit für r_{min}
- `M` Hyperflächen:
  - Klassenbezeichnung: `N` Bit
  - Radius: `N` bit
  - Referenzvektor: `D` Komponenten · `N` Bit

`r` hat somit eine Bitbreite von:

$$\Theta(\mathtt{r}) = \mathtt{N} + \mathtt{M}((2 + \mathtt{D}) \cdot \mathtt{N}) \tag{5.4}$$

Die Schaltung aus Abbildung 5.1 wurde willkürlich mit der Belegung `N:=8;D:=2;M:=4` instanziiert, wodurch die dargestellte Breite von $8+4\cdot((2+2)\cdot 8) = 136$ Bit zustande kommt.

5.2 Ausgänge

Nach der Klassifikation eines Musters liegt auf dem `N` Bit breiten Ausgang `class` die Bezeichnung der Klasse, welcher $\vec{x}$ zugeordnet wurde. Um im Falle einer Rückweisung den Grund angeben zu können, steht der Ausgang `aff` zur Verfügung und ist wie folgt zu interpretieren:

$$aff = \begin{cases} \mathtt{00}, & \text{wenn das Muster in keine Klasse fällt } (no); \\ \mathtt{01}, & \text{wenn das Muster in eine Klasse fällt } (one); \\ \mathtt{10}, & \text{wenn das Muster in mehrere Klassen fällt } (more) \end{cases} \tag{5.5}$$

Durch diese Fälle definiert sich auch der im Folgenden öfter verwendete Aufzählungstyp `AffType`

$$AffType := \{\mathtt{no = '00', one = '01', more = '10'}\} \tag{5.6}$$

der auch als *Klassifikationsstatus* bezeichnet wird. Im Falle einer Rückweisung - das Muster fällt in keine oder mehrere Klassen - wird eine Rückweisungsklasse benötigt. Diese ist durch das System festgelegt und wird mit der `N` Bit langen Null-Folge `'00...0'` am Ausgang `class` angezeigt. Es ist daher während der Bereitstellung der Trainingsdaten darauf zu achten, dass keine Klasse auf diese Weise bezeichnet wurde. Diese Konvention wurde eingeführt, da es Anwendungsfälle gibt, bei denen nicht zwischen den verschiedenen Varianten einer Rückweisungsursache differenziert werden muss. In solchen Situation kann der Ausgang `aff` des Klassifikators auch offen bleiben. Die Rückweisung ist dann am Auftreten der Rückweisungsklasse erkennbar.

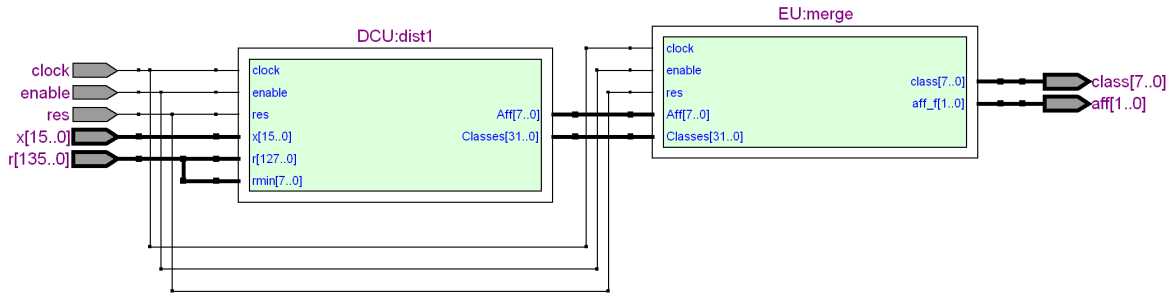


Abbildung 5.3: Grobe Struktur des Klassifikators

5.3 DCU und EU

Für die Realisierung des Klassifikationsalgorithmus aus Abschnitt 3.2 lassen sich zwei Kernelemente ableiten. Zunächst ist in einer *Distance Calculation Unit* (kurz DCU) für jede Hyperfläche zu entscheiden, ob das aktuelle Muster in deren Einflussbereich fällt. Da es möglich ist, dass Hyperflächen unterschiedlicher Klassen angesprochen werden, muss anschließend in einer *Evaluation Unit* (kurz EU) ausgewertet werden, welche Klassen durch die aktivierten Referenzvektoren in Frage kommen und wie dieses Ergebnis zu interpretieren ist. Abbildung 5.3 veranschaulicht die Struktur des Klassifikators.

Die **DCU** entscheidet für jede der Hyperflächen $h_l \in H$, ob $\vec{x}$ in deren Einflussbereich fällt und stellt das Ergebnis am Ausgang der **EU** zur Verfügung. Der Eingang des Klassifikators wird vollständig an die DCU weitergeleitet, nur r_{min} wird bereits auf einem separaten Signal geführt. Auf dem Ausgang `Classes` der DCU liegen die Klassenbezeichnungen $\tilde{h}_i$ der Hyperflächen. Wenn $H = \{h_{m-1}, h_{m-2}, \ldots, h_0\}$ die Menge der Hyperflächen ist, dann gilt:

$$\texttt{Classes'} = [\underline{\tilde{h}_{m-1}} \circ \underline{\tilde{h}_{m-2}} \circ \ldots \circ \underline{\tilde{h}_0}] \text{ und} \tag{5.7}$$

$$\texttt{Aff'} = [\underline{\lambda(h_{m-1})} \circ \underline{\lambda(h_{m-2})} \circ \ldots \circ \underline{\lambda(h_0)}] \tag{5.8}$$

Während `Classes` die zu jeder[1] Hyperfläche h_i gehörende Klassenbezeichnung $\tilde{h}_i$ enthält, ist der Leitung `Aff` zu entnehmen, ob eine Hyperfläche aktiviert wurde oder nicht. Eine Auskunft über die Hyperfläche selbst ist nicht mehr möglich. Es spielt auch keine Rolle, welche angesprochen wurde, da alleine aus der Belegung von `Classes` und `Aff` entschieden werden kann, ob und welche Klassen aktiviert wurden.

Das Resultat von λ (Gleichung 3.2) könnte zunächst mit einem Bit dargestellt werden. Eine Breite von `M` Bit würde unter diesem Aspekt für `Aff` ausreichen. Innerhalb der

[1] Es werden noch alle Hyperflächen aufgeführt, egal ob sie aktiviert wurden oder nicht.

EU wird neben diesen beiden Zuständen auch der Fall bedeutsam, dass $\vec{x}$ in mehreren Hyperflächen unterschiedlicher Klassen enthalten ist. Die EU vergleicht paarweise die Resultate von λ um auf eine Mehrfachzuweisung reagieren zu können:

$$\lambda(h_i) \wedge \lambda(h_j) \wedge (\tilde{h}_i \neq \tilde{h}_j) \Rightarrow \vec{x} \text{ wird durch mehrere Klassen erfasst} \tag{5.9}$$

Für diese Unterscheidung sind nun drei Zustände nötig. Damit innerhalb der EU die Daten nicht umformatiert werden müssen, wird der Ausgang `Aff` der DCU schon so konstruiert, dass er den Anforderungen der EU genügt. Es gilt $\Theta(\underline{\lambda(h_i)}) = 2$ und daher $\Theta(\texttt{Aff}') = 2M$.

Die EU erhält nun eine Menge von Klassenbezeichnungen und zugehörigen Aktivierungsresultaten. Schrittweise werden diese Angaben verglichen, wobei zunächst über je zwei der Hyperflächen folgende Entscheidung getroffen wird:

1. $\vec{x}$ wird durch keine Klasse erfasst.
2. $\vec{x}$ wird durch eine Klasse erfasst.
3. $\vec{x}$ wird durch mehrere Klassen erfasst.

Diese Ergebnisse werden schrittweise zusammengeführt, bis eine Aussage am Ausgang der EU erzeugt wird, die das Ergebis der Klassifikation ausmacht. Daher verfügt der Ausgang der EU auch nur noch über einen `N` Bit - Ausgang `class`, der den Namen der Klasse enthält und einen zwei Bit - Ausgang für den Klassifizierungsstatus (siehe Gleichung 5.5).

5.4 Die Distance Calculation Unit

Abbildung 5.4 zeigt eine Instanziierung der DCU für vier Hyperflächen. Für jede Hyperfläche wird eine *Activation Unit* (kurz AU) angelegt, die mit einer Hyperfläche h_i und dem Muster $\vec{x}$ versorgt wird. An dieser Stelle wird zum ersten Mal von der Methode Gebrauch gemacht, aus dem Konfigurationseingang `r` einzelne Hyperflächen zu extrahieren und je einer AU zuzuführen.

Entsprechend der Konstruktionsvorschrift von `r` lässt sich die Position einer Hyperfläche h_i in `r` wie folgt bestimmen:

- h_i ist ein Vektor $\vec{h}_i$, ein Radius $\mathring{h}_i$ und eine Klassenbezeichnung $\tilde{h}_i$
- der Vektor $\vec{h}_i$ besteht aus `D` Komponenten
- insgesamt also `D+2` Wörter der Länge `N`
- $\Theta(h) = (D+2)N$

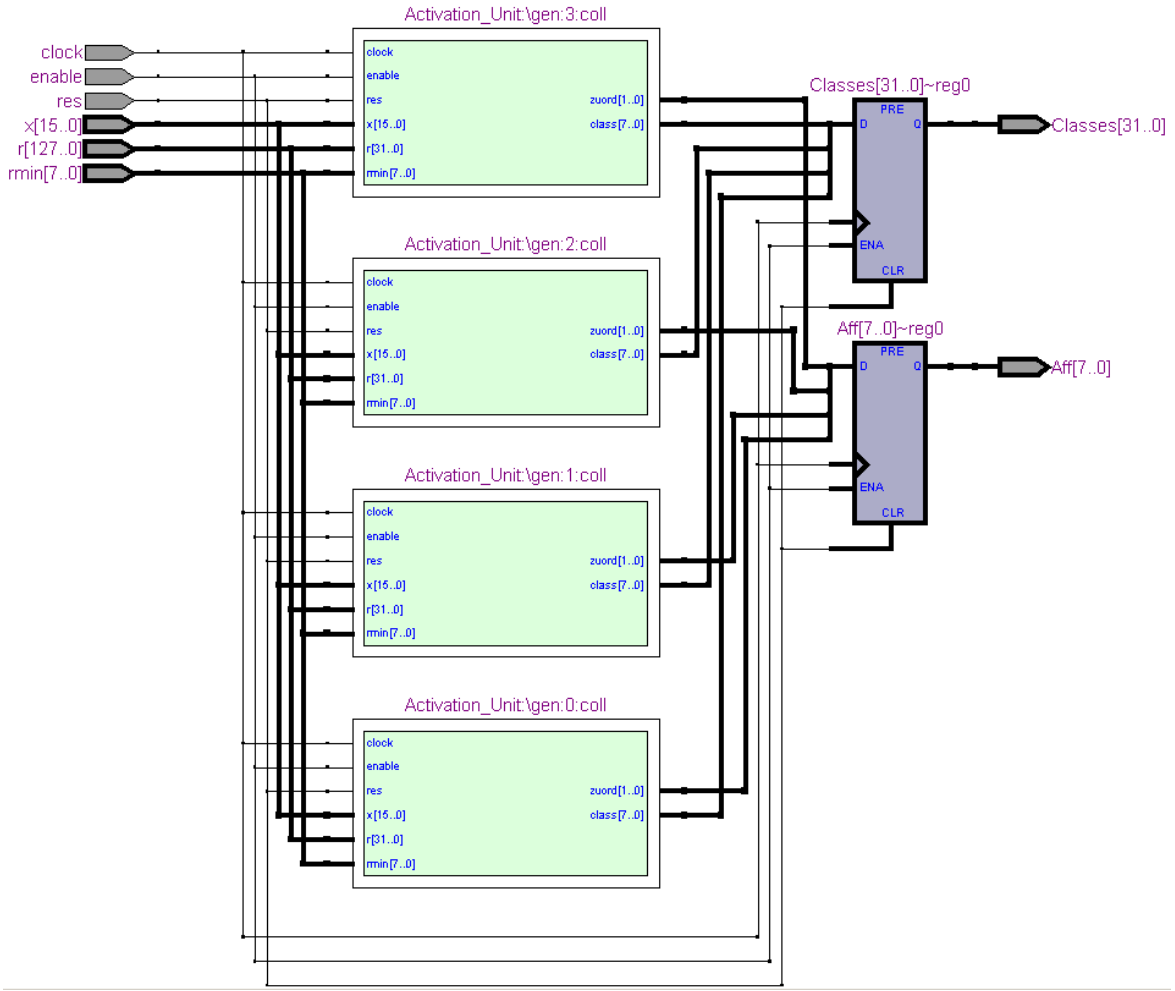


Abbildung 5.4: Die Distance Calculation Unit

h_i lässt sich aus `r` extrahieren durch:

$$\underline{h_i} = r[(i+1) \cdot \Theta(h) - 1 \ldots i \cdot \Theta(h)] \tag{5.10}$$

und wird dann der AU-Instanz AU_i zugewiesen. Auf ähnliche Weise wird das Resultat der einzelnen AUs wieder zusammengebaut, damit am Ausgang entsprechend der Betrachtung aus Abschnitt 5.3 die Ausgänge `Classes` und `Aff` die Ergebnisse aller AUs enthält. Hier zeigt sich der große Vorteil mit Generics zu arbeiten, da die Anzahl der AUs durch einen einzigen Parameter festgelegt werden kann und im Grunde nur durch die verfügbaren Ressourcen beschränkt ist.

5.4.1 Die Activation Unit

Für die Berechnung der Aktivierungsfunktion λ bestimmt die AU zunächst die Manhattan-Distanz $\delta(\vec{x}, \vec{h}_i)$ zwischen $\vec{x}$ und dem Referenzvektor $\vec{h}_i$. Der Ausgang `class` liefert die

Bezeichnung $\tilde{h}_i$ der Klasse ω_i, wenn $(\delta(\vec{x}, \vec{h}_i) < \mathring{h}_i) \wedge (\mathring{h}_i > r_{min})$ gilt. Der Ausgang `zuord` liefert den Klassifikationsstatus $\lambda(h_i)$.

5.4.2 Rekursive Berechnung der Manhattan-Distanz

Die Berechnung der Manhattan-Distanz muss derart erfolgen, dass sie für eine beliebige Anzahl von Merkmalen funktioniert. Da die Dimension des Merkmalsraums schon vor der Synthese durch den Parameter `D` angegeben wird, ist es möglich, eine Rekursive Struktur - ähnlich einem binären Baum - aufzubauen (siehe Abschnitt 4.3), die lediglich von `D` abhängig ist. Die Rekursion lässt sich wie folgt formulieren, wobei $\vec{x}^{\uparrow D}$ die obere Hälfte und $\vec{x}^{\downarrow D}$ die untere Hälfte der Dimensionen ist. $\vec{x}^1$ beschreibt den Fall, dass $\vec{x}$ eindimensional ist und als Skalar verarbeitet werden kann.

$$\delta(\vec{x}^1, \vec{h}^1) = |\vec{x}^1 - \vec{h}^1| \tag{5.11}$$

$$\delta(\vec{x}^D, \vec{h}^D) = \delta(\vec{x}^{\uparrow D}, \vec{h}^{\uparrow D}) + \delta(\vec{x}^{\downarrow D}, \vec{h}^{\downarrow D}) \text{ wenn } D > 1 \tag{5.12}$$

Für den Fall, dass während der Rekursion eine ungerade Dimension $D > 1$ auftritt, wird die Anzahl der Merkmale um ein neutrales Element erweitert, da die zweite Rekursionsgleichung eine gerade Anzahl von Merkmalen erwartet. Auf diese Weise entsteht die gewünschte Baumstruktur, deren Tiefe von der Anzahl der Merkmale abhängig ist. Die Tiefe trägt entscheidend zum Zeitverhalten der Schaltung bei, da an jedem inneren Knoten ein Register im Sinne des *Pipelings* angeordnet wird. Daraus folgt, dass pro Takt eine Addition (innere Knoten) oder der Absolutbetrag einer Differenz berechnet werden können muss. Näheres zur Bedeutung dieser Aussage ist Kapitel 7.3 zu entnehmen.

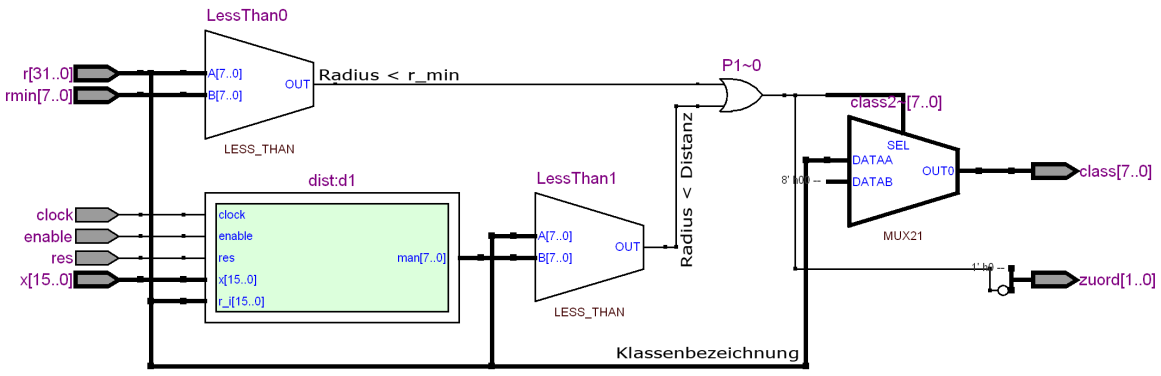


Abbildung 5.5: Architektur der Activation Unit

5.5 Die Evaluation Unit

Ähnlich wie die Berechnung der Manhattan-Distanz beruht auch die Auswertung der Aktivierungsfunktionen auf eine rekursive Struktur (Abbildung 5.6). Solange die Datenpfade `Classes` und `Aff` Information über mehr als zwei Hyperflächen führen, werden die Pfade in zwei Teile aufgespalten und separaten EUs zugeführt. Erst wenn

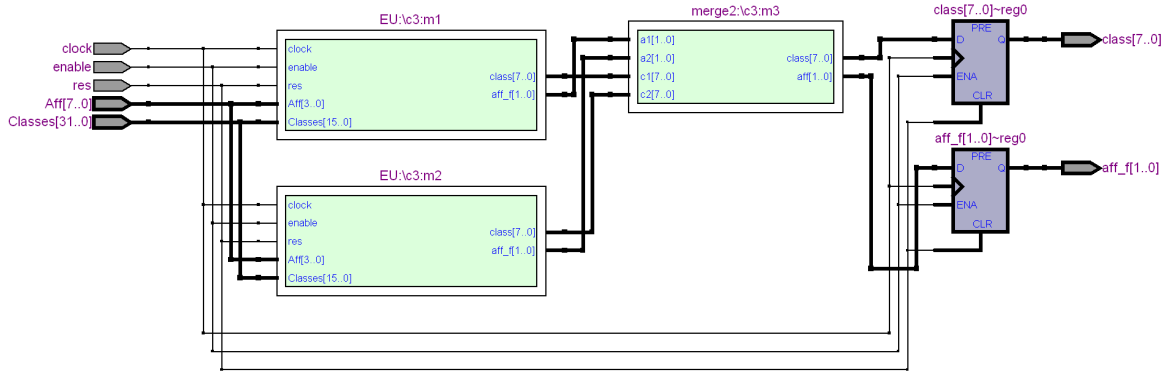


Abbildung 5.6: Architektur der Evaluation Unit

die Pfade nur noch zu genau zwei Hyperflächen Information bereitstellen, kann die Zuordnung des Musters ausgewertet werden. Sei

$$C^M = [\tilde{h}_{m-1} \circ \tilde{h}_{m-2} \circ \ldots \circ \tilde{h}_0] \text{ und} \tag{5.13}$$

$$A^M = [\lambda(h_{m-1}) \circ \lambda(h_{m-2}) \circ \ldots \circ \lambda(h_0)] \tag{5.14}$$

die initiale Belegung der Leitungen `Classes` und `Aff`. Die Rekursionsgleichung zur Bestimmung der entgültigen Klasse ist gegeben als:

$$\begin{aligned} \Psi(C^m, A^m) &= \psi(\Psi(C^{\uparrow m}, A^{\uparrow m}), \Psi(C^{\downarrow m}, A^{\downarrow m})) \; wenn \; m > 1 \\ \Psi(C^1, A^1) &= \psi(s,t) \; mit \; s = (\tilde{h}_1, \lambda(h_1)), \; t = (\tilde{h}_0, \lambda(h_0)) \end{aligned}$$

$$\psi\left((\tilde{h}_1, \lambda(h_1)), (\tilde{h}_0, \lambda(h_0))\right) = \begin{cases} (\tilde{h}_1, one) & \text{wenn } \lambda(h_1) = 1 \wedge \lambda(h_0) = 0 \\ (\tilde{h}_0, one) & \text{wenn } \lambda(h_1) = 0 \wedge \lambda(h_0) = 1 \\ (\tilde{h}_1, one) & \text{wenn } \lambda(h_1) = 1 \wedge \lambda(h_0) = 1 \wedge \tilde{h}_1 = \tilde{h}_0 \\ ('00..0', no) & \text{wenn } \lambda(h_1) = 0 \wedge \lambda(h_0) = 0 \\ ('00..0', more) & \text{sonst.} \end{cases}$$

6

Zeitverhalten und Platzbedarf

Der Klassifikator selbst gibt am Ausgang nur noch das Klassifikationsergebnis durch Angabe der erkannten Klasse und des Klassifizierungsstatus (siehe Gleichung 3.2) aus. Welches Muster verarbeitet wurde, ist ohne Weiteres nicht ersichtlich. Um beispielsweise das Muster auf einer separaten Pipeline vorbei zu schleusen, muss die Anzahl der Pipelinestufen bekannt sein. Dieser Wert wird auch als *Latenz* bezeichnet und beschreibt die Verzögerung der Schaltung.

Wie bereits in Abschnitt 5.1 beschrieben wurde, besteht die Schaltung aus Logikbausteinen, die über Register verbunden sind. Ein am Eingang anliegendes Datenwort springt nun bei jeder Taktflanke von einem Register zum nächsten, und erst nachdem alle Register durchlaufen sind, liegt das Resultat am Ausgang an. Es bleibt also zu klären, wie die Pipelinetiefe in Abhängigkeit der Parameter N, D und M variiert.

Für den praktischen Einsatz ist weiterhin interessant, wie der Platzbedarf von diesen Parametern abhängt. Es muss gewährleistet sein, dass eine entwickelte Architektur auf der gewählten Hardware den nötigen Platz findet.

Mit Hilfe der in Abschnitt 5.1 vorgestellten Generics wird der Klassifikator individuell synthetisiert, wodurch sich das zeitliche Verhalten und der Platzbedarf ändern. Im Folgenden wird untersucht, wie sich die einzelnen Parameter auf die Schaltung auswirken, damit eine Formel für die Verzögerung der Schaltung angeben werden kann. Die

asymptotische[1] Komplexität wird hierbei mit diskutiert.

6.1 Analyse einzelner Komponenten

6.1.1 Distance Calculation Unit

Abbildung 5.4 der *Distance Calculation Unit* zeigt, dass unabhängig von den Parametern das Resultat der *Activation Units* gebündelt in zwei Register geschrieben wird. Diese Register existieren also in jeder Variante des Klassifikators.

Eine dynamische Anzahl von Pipelinestufen entsteht u.a. durch die Konstruktion des binären Baumes zur Berechnung der Manhattan-Distanz. Register werden hier eingesetzt um die Distanz zwischen zwei Merkmalen eines Musters und einem Referenzvektor zu bestimmen und um die Addition zweier Merkmale zu speichern. Insgesamt

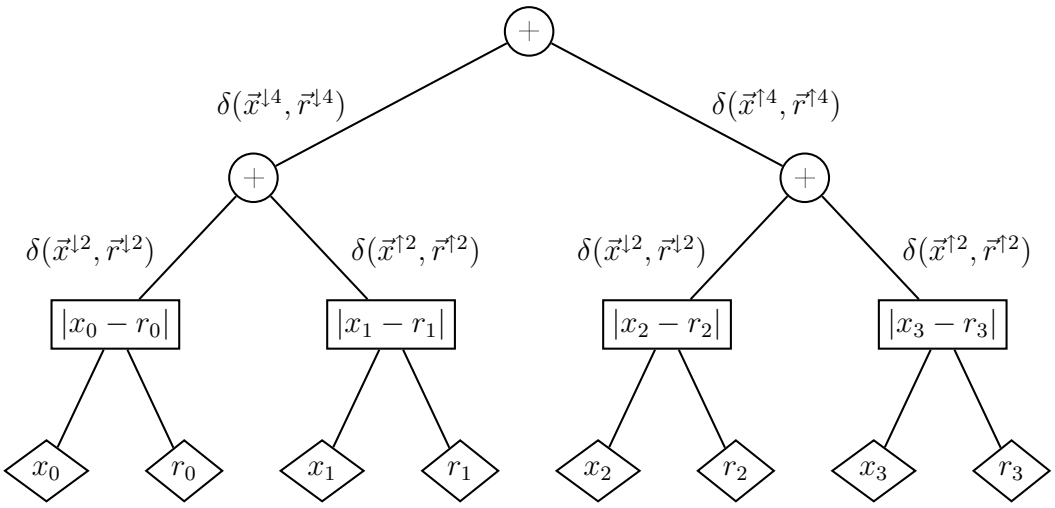


Abbildung 6.1: Baumstruktur der Manhattan-Distanz-Berechnung

müssen `D` Teilabstände aufsummiert werden. Die Komplexität der Distanzberechnung ergibt sich also direkt aus dem Parameter `D`, und der Gestalt des daraus entstehenden Baumes:

$$\mathcal{O}_{time}(\delta(\vec{x}, \vec{r})) = \mathcal{O}(\log_2 D) \tag{6.1}$$

$$\mathcal{O}_{area}(\delta(\vec{x}, \vec{r})) = \mathcal{O}(D) \tag{6.2}$$

Es sei noch gesagt, dass der Platzbedarf $\mathcal{O}_{area}$ für die Distanzberechnung M mal so hoch ist, da für jede Hyperfläche eine separate Einheit angelegt wird.

[1] abstrakte Komplexität, wenn die Parameter gegen ∞ laufen

Latenz der DCU

Unter Berücksichtigung des konstanten Registers und der logarithmischen Entwicklung in Abhängigkeit von `D` ergibt sich eine Latenz von:

$$Latenz(DCU) = \lceil \log_2 2D \rceil + 1 = 2 + \lceil \log_2 D \rceil \tag{6.3}$$

6.1.2 Activation Unit

Die Analyse der AU bringt eine ähnliche Komplexität zum Vorschein. Es existiert lediglich kein konstantes Register, das berücksichtigt werden muss. Die Rekursion generiert einen binären Baum, der M - Blätter enthält, um die M Hyperflächen miteinander vergleichen zu können. Der Wurzelknoten als auch alle inneren Knoten entsprechen einer AU, die am Ausgang das Ergebnis in Registern speichert (siehe Abbildung 5.6):

$$\mathcal{O}_{time}(AU) = \mathcal{O}(\log_2 M) \tag{6.4}$$
$$\mathcal{O}_{area}(AU) = \mathcal{O}(M) \tag{6.5}$$

Die Latenz liegt bei:

$$Latenz(EU) = \lceil \log_2 M \rceil \tag{6.6}$$

6.1.3 Zusammenfassung

Diese Implementierung des Klassifikators bringt den großen Vorteil der logarithmischen Schaltungstiefe mit sich. Im Sinne einer möglichst schnellen Klassifikationsdurchführung ist das die beste Lösung, was positiv zu bewerten ist. Dieses Optimum Bedingt linearen Platzbedarf, was in der Praxis schnell zu kritischen Situationen führen kann.

Gesamtlatenz

Die Latenz des gesamten Klassifikator-Moduls ergibt sich in Abhängigkeit der beiden Parameter `M` und `D` und den Betrachtungen der Abschnitte 6.1.1 und 6.1.2 aus

1. einem festen Register in der DCU,
2. $log_2(2D)$ Stufen während der Abstandsberechnung und
3. $log_2(M)$ Stufen innerhalb der AU.

Das System hat also insgesamt eine Latenz von:

$$Latenz(Klassifikator) = 2 + \lceil \log_2 D \rceil + \lceil \log_2 M \rceil \tag{6.7}$$

Der Parameter `N` hat keinen Einfluss auf die Schaltungstiefe, da er nur die Bitbreite der Datenpfade bestimmt. Sein Einfluss auf das zeitliche Verhalten wird erst dann interessant, wenn wegen der hohen Auflösung die Bearbeitungszeit eines Taktes nicht ausreicht, um das Resultat einzelner Pipelinestufen zu bestimmen. Diese Thema wird in Abschnitt 7.3 diskutiert.

7 Ergebnisse

Der aus den zuvor geführten Betrachtungen resultierende Platzbedarf soll nun experimentell belegt werden. Dazu wird das Layout mit unterschiedlichen Werten für die Parameter `D` und `M` synthetisiert. Der Parameter `N` für die Bitbreite wird jedoch festgehalten. Auch wenn die Wahl von `N` den Platzbedarf mitbestimmt, da alle Register mit steigenden Werten auch größer werden, soll dieser Parameter beim Platzbedarf nicht berücksichtigt werden. Für `N` wird ein Wert festgelegt, der in Hinblick auf den pratischen Einsatz bei der Verarbeitung von Sensordaten realistisch erscheint.

Die Bedeutung der Bitbreite für das Zeitverhalten spiegelt sich vor allem in der maximalen Frequenz wieder, mit der die Schaltung betrieben werden darf. In den kleinsten Verarbeitungsstufen, die nicht durch zusätzliche Register im Sinne des Pipelinings aufgeteilt sind, finden elementare Operationen wie Logik, Vergleiche oder Additionen statt. Mit steigender Bitbreite verschlechtert sich das zeitliche Verhalten dieser Stufen, da mehr Information in der gleichen Zeit verarbeitet werden muss. Daraus resultiert, dass für eine gewählte Bitbreite eine maximale Taktrate existiert, die gerade noch ausreicht, um in einem Takt die Operation auszuführen.

7.1 Der Klassifikator des NI1000 - Neurochip

Der NI1000 ermöglicht die Verarbeitung von **256-dimensionalen** Vektoren, wobei die einzelnen Werte mit **fünf Bit** aufgelöst werden. Insgesamt kann zwischen **64** verschiedenen **Klassen** separiert werden. Für die Beschreibung des Merkmalsraums verfügt

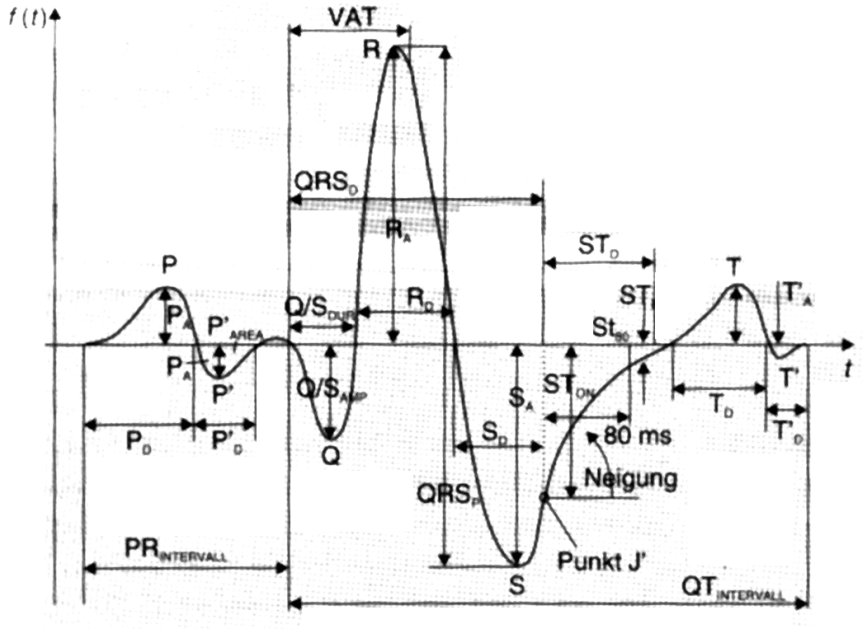


Abbildung 7.1: Kenngrößen in einem EKG(Quelle: [MH04])

der NI1000 über einen Speicher, in dem die Hyperflächen gespeichert werden. Die Distanzberechnung findet in serieller Weise statt, wie sie in Kapitel 4 beschrieben wurde. Die Klassifizierung von 33.000 Mustern pro Sekunde wird erst dadurch möglich, dass 512 solcher Berechnungsmoduln existieren [Nes95].

Konfiguration für die Feuererkennung

Für den **Einsatz in der Feuererkennung** auf dem Kleinstsatelliten Bird dienen drei Spektralbilder als Eingangsgrößen, wodurch der resultierende Merkmalsraum drei Dimensionen hat. Mit **60 Hyperflächen** konnte der Merkmalsraum entsprechend den Lerndaten beschrieben werden.

Die Möglichkeiten des NI1000 werden für die Feuererkennung nur in Bruchteilen genutzt. Ein Merkmalsraum mit 256 Dimensionen wird erst dann interessant, wenn die zu klassifizierenden Objekte auch entsprechend viele Merkmale haben. Kardiologen beispielsweise können einem EKG bis zu 30 Kenngrößen entnehmen, deren Ausprägung Aufschluss über pathologische Vorgänge liefert (siehe Abb. 7.1).

7.2 Experimentelle Ermittlung des Platzbedarfs

Im Folgenden soll der Platzverbrauch des Klassifikators experimentell bestätigt werden. Dadurch kann die theoretische Annahme über das Platzverhalten validiert werden. Für

diesen Test wird wieder der Cyclone II zu Grunde gelegt, die mit bis zu 68.000 LEs zu erhalten ist. Bei der Synthese wird trotzdem versucht, Werte von 330.000 LEs zu erreichen, da die 68.000 noch immer im unteren Sektor der Altera-Produkte liegen. Die meisten LEs (340.000 LEs) sind derzeit bei Altera auf einem **Stratix III EP3SL340** zu finden.

Altera vs. Xilinx

Xilinxs Äquivalent für die *Logic Elements* sind die *Logic Cells* (LCs). Die neueste Produktreihe von Xilinx - der **Virtex V** - ist mit bis zu 330.000 LCs zu erhalten. LEs und LCs können nicht direkt miteinander verglichen werden. Sie sind auch nur bedingt ein Maß für die Leistung. Ein FPGA mit weniger LEs, aber etwas mehr eingebetteten Multiplizierern, kann um einiges stärker sein. Neben den LEs sind prinzipiell alle on-Chip-Ressourcen für eine objektive Leistungseinschätzung notwendig.

Beide *Maßeinheiten*, LE und LC, beschreiben eine LUT mit vier Eingängen und ein Register [Wan98] [Alt04]. Wie in Abb. 2.10 erkennbar ist, sind neben diesen Bestandteilen noch weitere Elemente in einem Logikblock untergebracht. Diese *Zusatzlogik* kann je nach Hersteller und Produkt anders ausfallen. Generell kann also nicht 1:1 zwischen LEs und LCs umgerechnet werden. Trotzdem kann das resultierende Ergebnis der folgenden Analyse *in etwa* für beide Hersteller angewendet werden.

Festlegung einer Bitbreite

Mit dem gewählten Entwurf des Klassifikators kann zunächst jede beliebige Bitbreite genutzt werden. Damit wurde bereits eines der wesentlichen Ziele dieser Arbeit erreicht. Für den folgenden Test wird unter Berücksichtigung praxisrelevanter Auflösungen eine Bitbreite von **16 Bit** festgelegt.

Syntheseergebnisse

Der gesamte Prozess wurde mit dem Entwicklungstool **Quartus II 7.0** (siehe [Alt07]) von Altera durchgeführt. Dieses Werkzeug ist, ähnlich wie **ISE** von Xilinx, eine Sammlung vielfältiger Tools, die den kompletten Entwicklungszyklus unterstützen. Neben den integrierten Tools für die Synthese, Timinganalyse und Simulation besteht die Möglichkeit, Drittanbieterprodukte einzubinden. Für die folgende Synthese wurde ausschließlich Quartus II Software genutzt.

Abbildung 7.2 veranschaulicht, wie viele LEs in Anbhängigkeit von `M` und `D` bei einer Auflösung von 16 Bit auf einem Cyclone II benötigt werden. Tabelle 7.2 zeigt

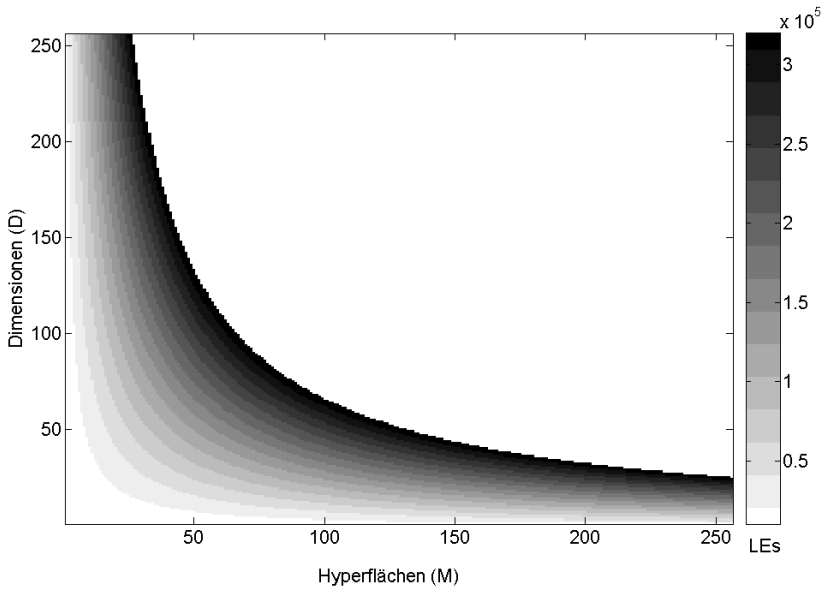


Abbildung 7.2: Platzbedarf auf Cyclone II bei 16 Bit Auflösung

die experimentell ermittelten Werte, aus denen die Werte in Grafik 7.2 hervorgegangen sind. Vor allem der lineare Zusammenhang zwischen Platzbedarf und den beiden Parametern ist deutlich zu erkennen.

7.3 Die Maximale Frequenz

Die maximale Frequenz f_{max} ist eine Größe, die angibt, mit welcher Taktrate eine bestimmte Schaltung höchstens laufen darf, damit das gewünschte Verhalten gewährleistet werden kann. Die Parameter `D` und `M` haben auf diesen Wert keinen Einfluss, da sie sich vorwiegend auf den Platzverbrauch auswirken. Es bleibt zu berücksichtigen, dass mit zunehmender Größe des Layouts die Pfade zwischen einzelnen Komponenten größer werden und daher ein Zusammenhang zwischen `M`, `D` und f_{max} bestehen kann.

Ein Layout kann durch ein *Timing-Analyse-Tool* auf diesen Wert hin untersucht werden. Ermittelt wird er hauptsächlich aus der größten *Register-zu-Register-Verzögerung*. Dieser Wert gibt an, wie lange es dauert, bis ein in einem Register gespeicherter Wert das nächste erreicht. Er hängt im wesentlichen von der Logik ab, die zwischen diesen Registern liegt. Für nähere Hinweise zur Bestimmung von f_{max} und den verwendeten Werkzeugen sei auf [Alt07] verwiesen. Der Klassifikator wurde mit Auflösungen zwischen 2 und 64 Bit erstellt. Die resultierenden Frequenzen f_{max} sind in Abbildung 7.3 dargestellt.

D-M	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**
2	298	633	1.300	2.600	5.300	10.600	21.400	42.900	85.800
4	490	1.000	2.000	4.200	8.400	16.900	34.000	67.000	135.000
8	886	1.800	3.600	7.300	14.700	29.500	59.000	118.100	236.240
16	1.600	3.300	6.700	13.600	27.200	54.600	109.200	219.000	438.110
32	3.200	6.500	13.00	26.100	52.300	104.700	209.000	418.070	-
64	6.300	12.700	25.600	51.200	102.500	204.100	408.000	-	-
128	12.640	25.320	50.690	101.400	202.800	405.640	-	-	-
256	25.200	50.400	100.000	200.000	399.000	-	-	-	-
512	50.000	100.000	200.000	399.600	-	-	-	-	-

Tabelle 7.1: Synthetisierungsergebnisse mit unterschiedlichen Parametern

7.4 Fazit

Vergleicht man die Testergebnisse mit den Leistungen des NI1000, so ist vor allem das zeitliche Verhalten auffällig. Während der Neurochip 33.000 Klassifizierungen/Sekunde ermöglicht, schafft der gewählte Entwurf mit einer Auflösung von 6 Bit noch 230 Millionen Klassifizierungen/Sekunde. Die für den Test festgelegte Bitbreite von 16 Bit ermöglicht immer noch 150 Millionen Klassifizierungen/Sekunde. Diese Leistung ist vor allem auf die Realisierung nach dem Prinzip des Pipelinings zurückzuführen. Nicht zuletzt wird diese Leistung durch die parallele Berechnung von Teilergebnissen in Form einer binären Baumstruktur von Verarbeitungseinheiten ermöglicht, deren Anzahl von den gewählten Parametern abhängig ist. Daraus resultiert auch der linear von den Parametern `D` und `M` abhängende Platzbedarf.

Der benötigte Platz auf einem Cyclone II reicht im Grunde nicht, um die Möglichkeiten des NI1000 anzunähern. Ein direkter Vergleich zwischen beiden Varianten erscheint

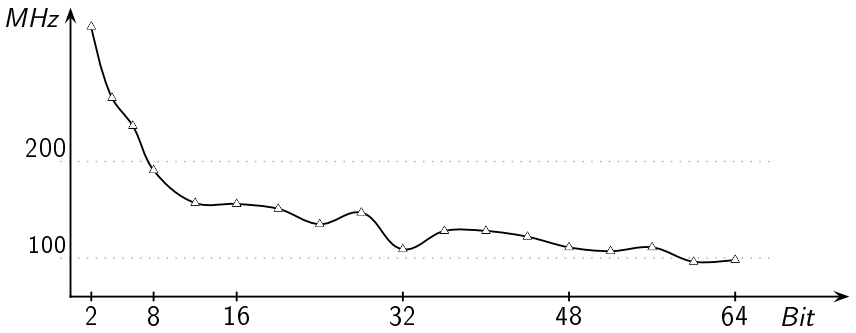


Abbildung 7.3: Abhängigkeit der maximalen Frequenz von der Bitbreite

dennoch nicht sinnvoll, da die gewählte Technologie nur exemplarisch verwendet wurde. Grundsätzlich ist dem NI1000 die Eigenschaft gutzuschreiben, die Hyperflächen in einem Speicher abzulegen. Ein entsprechender VHDL-Entwurf würde in seiner Platz-Komplexität nicht von den Parametern `D` und `M`, sondern nur vom verfügbaren Speicher abhängen. Dann existiert nur eine bestimmte Anzahl von Verarbeitungseinheiten, die alle im Speicher liegenden Hyperflächen nacheinander verarbeiten. Auf diese Weise würde sich das zeitliche Verhalten aber wieder verschlechtern, sobald mehr Hyperflächen als Verarbeitungseinheiten existieren.

Es zeigt sich aber, dass die verwendete Konfiguration des NI1000 für die Feuererkennung (N=5;D=3;M=60) auf dem Cyclone II unter Verwendung von 4.400 LEs realisiert werden kann. Das sind 6.5% der verfügbaren Ressourcen auf einem Cyclone II mit 68.000 LEs. Bei 16 Bit Auflösung werden bereits 20% verwendet.

8

Implementierung des Trainingsalgorithmus

In Abschnitt 1 wurde bereits erwähnt, dass der Trainingsalgorithmus nicht in Hardware, sondern durch Software realisiert wird. Sinnvoll ist diese Entscheidung vor allem, weil das Training nicht unter Echtzeitbedingungen stattfindet. Im Gegensatz zur Klassifizierung muss während der Trainingsphase kein Strom von Daten verarbeitet werden, da die Trainingsdaten vor dem Training bereits vorliegen. Weiterhin muss das Resultat des Trainings nicht innerhalb einer kritischen Zeit bereitstehen, so dass die Vorteile einer Hardwareimplementierung, schnell und massiv parallel zu sein, hier nicht notwendig sind.

Wenn Algorithmen in Hardware implementiert werden, spielt die Komplexität der Teiloperationen stets eine wichtige Rolle, da bestimmte Operationen in Bezug auf Platz- und Zeitverhalten sehr teuer sein können. Die Qualität der Merkmalsraumaufteilung sollte nicht von solchen Überlegungen abhängen, und es sollten solche Operationen und Verfahren gewählt werden, die eine möglichst optimale Merkmalsraumaufteilung liefern. Der Kompromiss zwischen Rechenaufwand und Leistung sollte beim Training zu Gunsten der Leistung ausfallen. Wird dieser sehr wichtige Teil der gesamten RCE-Netzwerk-Klassifikation durch Software realisiert, kann die Größe des Hardwarelayouts deutlich minimiert werden.

Es sind also die folgenden Argumente, die eine Implementierung in Software rechtfertigen:

1. Es müssen keine Daten in einer bestimmten Zeit verarbeitet werden.
2. Der Klassifikator muss nur einmal im Vorfeld seiner eigentlichen Arbeit trainiert werden.
3. Der Platzbedarf der Schaltung kann durch eine Software-Implementierung des Trainings wesentlich verkleinert werden.

Der Trainingsalgorithmus muss aus der Trainingsdatenmenge T die Merkmalsraumaufteilung H bestimmen und entsprechend den Anforderungen der Konfigurationsleitung (s. S. 33) codieren. Resultat ist also ein Datenwort, das an die Konfigurationsleitung angelegt werden kann.

In diesem Abschnitt werden zunächst einige Erweiterungen des Algorithmus vorgestellt, die zur Effizienzsteigerung beitragen sollen, **ohne** die Funktionsweise des Verfahrens grundlegend zu ändern. Die Implementierung wird nur kurz vorgestellt, da sie dem Algorithmus aus Abschnitt 3.1 entspricht.

8.1 Optimierung der Algorithmik

8.1.1 Eingeschlossene Hyperflächen

Da die verfügbaren Ressourcen in jedem Fall beschränkt sind, ist es sinnvoll, für ein Klassifizierungsproblem eine minimale Menge von Hyperflächen zu finden. Der Trainingsalgorithmus trifft keine Aussage über die Reihenfolge, mit der die einzelnen Trainingsvektoren verarbeitet werden. Er findet bei jeder Permutation der Trainingsdaten eine geeignete Aufteilung, nur dass die Anzahl und Gestalt der Hyperflächen variieren kann.

Ein Beispiel

Für eine gegebene Menge von Trainingsdaten ist es denkbar, dass eine Permutation aus jedem Trainingsvektor eine Hyperfläche konstruiert (*ungünstigster Fall*), und eine andere deren Trainingsresultat für jede Klasse genau eine Hyperfläche erstellt (*günstigster Fall*). Ein Beispiel für den ungünstigen Fall zeigt Abbildung 8.1.

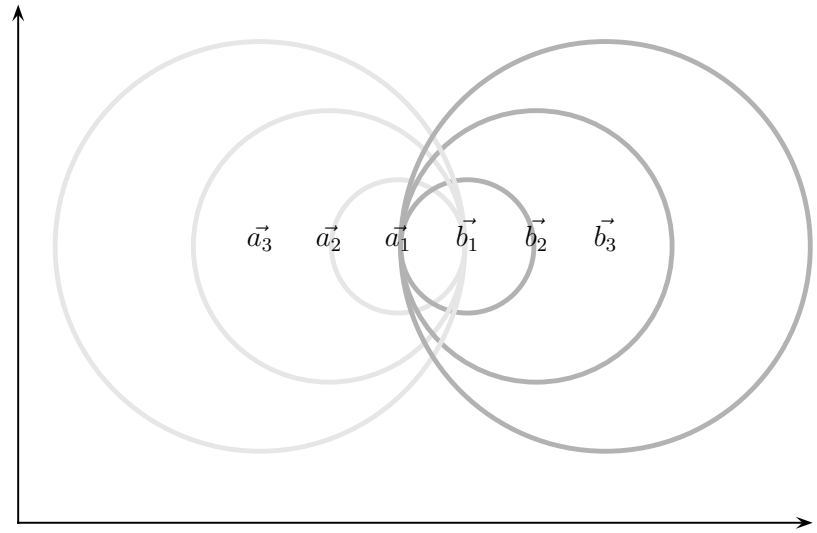


Abbildung 8.1: Eingeschlossene Hyperflächen

Die Trainingsdaten wurden in der Reihenfolge $\vec{a}_1, \vec{b}_1, \vec{a}_2, \vec{b}_2, \vec{a}_3, \vec{b}_3$ verarbeitet, wobei r_{min} minimal und r_{max} sehr groß gewählt wurde. Nachdem im ersten Schritt $\vec{a}_1$ als Hyperfläche aufgenommen wurde, fällt im zweiten Schritt $\vec{b}_1$ in dessen Einflussbereich, was dazu führt, dass die Radien beider Flächen auf deren Abstand gesetzt werden. Nun erfassen die Hyperflächen nicht mehr die fremdklassigen Vektoren. Wird $\vec{a}_2$ verarbeitet, fällt er zunächst in eine unbeschriebene Region. Er fällt zwar auf den Rand der Hyperfläche um $\vec{a}_1$, aber das reicht nicht aus, da er echt innerhalb der Fläche liegen muss. Es reicht auch keine fremde Hyperfläche bis zu $\vec{a}_2$, wodurch eine neue Hyperfläche mit Radius r_{max} angelegt wird. Erst wenn $\vec{b}_2$ einsortiert wird und dabei in den Einflussbereich von $\vec{a}_2$ fällt, verringern sich die Radien beider Vektoren. Nachdem alle Vektoren einsortiert wurden, reicht $\vec{a}_1$ bis an $\vec{b}_1$, $\vec{a}_2$ an $\vec{b}_2$ und letztlich $\vec{a}_3$ bis an $\vec{b}_3$. Abbildung 8.2 zeigt den Zustand der Merkmalsraumaufteilung nach dem ersten Durchlauf.

Erst wenn in einer zweiten Iteration $\vec{a}_1$ erneut einsortiert wird und in die Hyperflächen um $\vec{b}_2$ und $\vec{b}_3$ fällt, verändert sich deren Radius auf die endgültige Größe (analog für $\vec{b}_1$). Der finalen Konfiguration entspricht die Darstellung aus Abbildung 8.1, bei der kein Vektor mehr durch eine fremde Hyperfläche erfasst wird.

Aus Abbildung 8.1 geht zum einen hervor, dass die Reihenfolge, in der die Trainingsdaten verarbeitet wurden, die ungünstigste war, da die Anzahl der Hyperflächen der Anzahl der Trainingsvektoren entspricht. Andererseits zeigt sich, dass, obwohl die Klassenregionen praktisch überlappungsfrei existieren könnten, die Hyperflächen Schnittregionen haben und das, obwohl r_{min} in keinem Fall erreicht wurde. Es zeigt sich, dass entgegen [Hal99] von $\mathring{h}_i > r_{min}$ für alle i **nicht** auf eine überlappungsfreie Merkmalsraumaufteilung geschlossen werden kann.

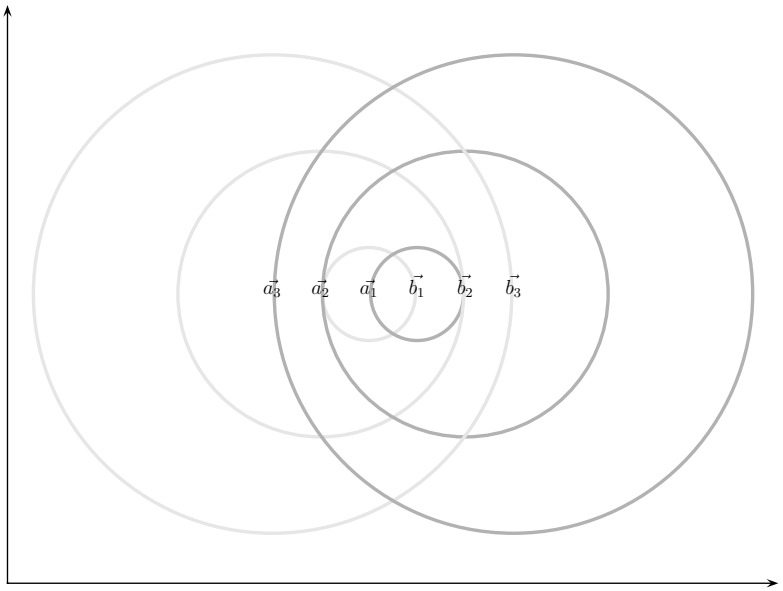


Abbildung 8.2: Gestalt der Hyperflächen nach einer Iteration

Vermeidung eingeschlossener Hyperflächen

Wenn $h_i \subset h_l$ gilt, also h_i echte Teilmenge von h_l ist, werden alle Muster in h_i auch durch h_l erfasst. Da $\mathring{h}_l > \mathring{h}_i$ muss $\mathring{h}_l$ größer als r_{min} sein, da der kleinst mögliche Radius r_{min} ist (siehe Seite 22). Wird nun während der Trainingsphase h_l vor h_i einsortiert, muss h_l wieder den Radius $\mathring{h}_l$ erhalten, da $\mathring{h}_l > r_{min}$ bedeutet, dass bei diesem Radius keine Konflikte mit fremdklassigen Referenzvektoren auftreten. Bei Konflikten würde $\mathring{h}_l$ verkleinert werden bis maximal r_{min}. Für die Region h_i wird nun keine Hyperfläche mehr angelegt, da diese bereits durch h_l erfasst wird.

Daraus lässt sich folgern, dass die Anzahl der Hyperflächen verringert werden kann, wenn das Training mit den Trainingsvektoren beginnt, die nach Abschluss des Trainings den größten Radius haben. Neben den eingeschlossenen Hyperflächen verschwinden so auch solche Hyperflächen, die ihren Mittelpunkt in einer größeren Hyperfläche der eigenen Klasse haben.

Der Algorithmus wird nun derart erweitert, dass nach Abschluss der Trainingsphase die Hyperflächen der Größe nach absteigend sortiert werden. Mit der daraus entste-

henden neuen Reihenfolge der Trainingsdaten wird der Algorithmus erneut ausgeführt, wodurch das Auftreten von eingeschlossenen Hyperflächen vermieden wird.

Diese Erweiterung ist gewissermaßen keine Veränderung, da das Verfahren prinzipiell so geschaffen ist, dass die Reihenfolge keine Rolle spielt. Die im Beispiel gezeigten

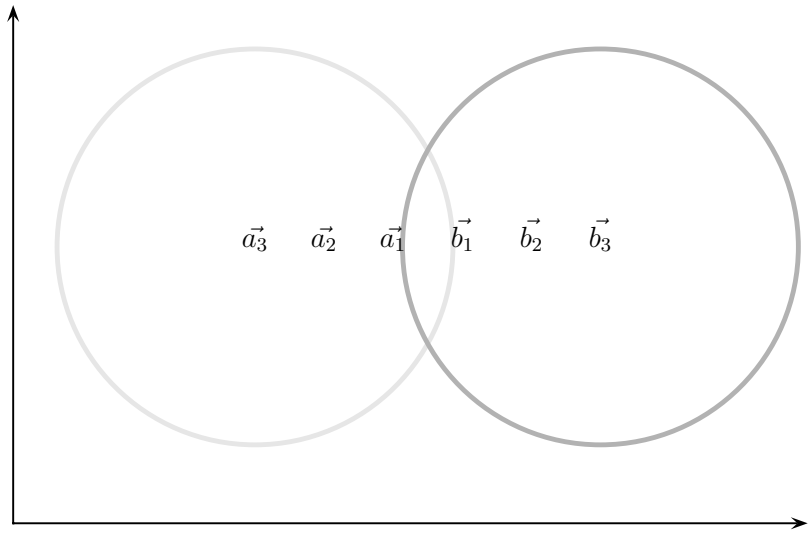


Abbildung 8.3: Hyperflächen, nach veränderter Reihenfolge

Trainingsdaten führen dann zu der in Abbildung 8.3 gezeigten Merkmalsraumaufteilung. Hier liegt jetzt der günstigste Fall vor, da für jede Klasse nur eine Hyperfläche ermittelt wurde.

8.1.2 Reduktion von Überlappungen

Nach wie vor zeigt die Aufteilung aus Abbildung 8.3 eine Überlappung, welche durch reine Permutation der Trainingsdaten nicht beseitigt werden kann.

Das liegt vor allem daran, dass die Radienverkleinerung vom Vergleich der Trainingsvektoren mit den Hyperflächen abhängig sind. Es werden **nicht** die Hyperflächen untereinander verglichen, wodurch Überlappungen erst sichtbar werden.

Überlappung werden zunächst nur dann erkannt, wenn ein Trainingsdatenvektor in diese Region fällt. Alle fremdklassigen Hyperflächen werden dann so lange verkleinert, bis sie diesen Punkt nicht länger erfassen. Die Überlappung wird dadurch unter Umständen nicht beseitigt. Es kann keine Aussage darüber getroffen werden, ob die Überlappung völlig verschwunden ist.

Es sei erwähnt, dass diese Dichte der Klassen eher theoretischer Natur ist und in der Praxis gemieden wird. Führen die gewählten Merkmale zu einem Mermalsraum, in

dem die unterschiedlichen Klassen so dicht beieinander liegen, sollten andere Merkmale gewählt werden, die eine bessere Aufteilung ermöglichen.

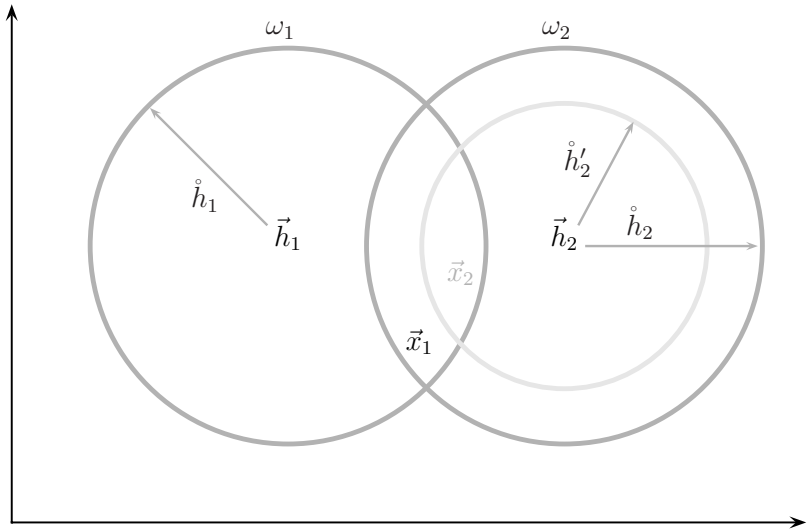


Abbildung 8.4: Überlappungen

Abbildung 8.5 veranschaulicht, wie eine Überlappung zwar verkleinert, jedoch nicht eliminiert werden kann. Der Merkmalsraum wird bereits durch zwei Hyperflächen h_1 und h_2 (mit den Radien $\mathring{h}_1$ und $\mathring{h}_2$) beschrieben, die unterschiedlichen Klassen ω_1 und ω_2 angehören und eine Überlappung haben. Der Trainingsvektor $\vec{x}$, als Vertreter der Klasse ω_1, fällt in die Überlappung, wodurch der Radius der Hyperfläche h_2 von $\mathring{h}_2$ auf $\mathring{h}'_2$ verkleinert wird. $\vec{x}_1$ wird nun durch h_2 nicht länger erfasst. Und der Vorgang der Merkmalsraumaufteilung ist abgeschlossen. Hier zeigt sich, dass durch die Veränderung der Radien die Überlappung zwar verkleinert, aber nicht beseitigt werden konnte, da zum Beispiel der Punkt $\vec{x}_2$ in der verbleibenden Überlappung liegt.

Hyperflächen unterschiedlicher Klassen überlappen, wenn die Summe ihrer Radien größer als der Abstand ihrer Referenzvektoren ist. Diese Erkenntnis kann genutzt werden, um die Menge der Hyperflächen auf Überlappungen zu prüfen.

Der Algorithmus wird nun derart erweitert, dass nach Abschluss der Trainingsphase die Hyperflächen paarweise verglichen werden. Wird dabei eine Überlappung registriert, werden beide Hyperflächen so weit verkleinert, bis sie nicht mehr überlappen. Die Überlappung ist beseitigt, wenn die Summe der Radien $\mathring{h}_i$ und $\mathring{h}_j$ der Distanz $\delta(\vec{h}_i, \vec{h}_j)$ entspricht. Dann ragen die beiden Hyperflächen genau aneinander. Da die Hyperflächen unterschiedlich groß sein können, sollte die Verkleinerung fair erfolgen. Fair bedeutet hier, dass große Hyperflächen mehr und kleine weniger verkleinert werden. Dazu werden die Radien in dem Verhältnis verkleinert, in dem auch die Radien zueinander stehen.

Seien h_1 und h_2 zwei überlappende Hyperflächen:

$$\tilde{h}_1 \neq \tilde{h}_2 \wedge \delta(\vec{h}_1 + \vec{h}_2) < \mathring{h}_1 + \mathring{h}_2 \tag{8.1}$$

Die Größe u der Überlappung ist die Differenz zwischen der Summe der Radien und dem Abstand der beiden Vektoren:

$$u = (\mathring{h}_1 + \mathring{h}_2) - \delta(\vec{h}_1 + \vec{h}_2) \tag{8.2}$$

Der neue Radius $\mathring{h}_{1,neu}$ für die Hyperfläche h_1 ergibt sich wie folgt:

$$\mathring{h}_{1,neu} = \mathring{h}_1 - \left(u \cdot \frac{\mathring{h}_1}{\mathring{h}_1 + \mathring{h}_2} \right) \tag{8.3}$$

und für h_2:

$$\mathring{h}_{2,neu} = \mathring{h}_2 - \left(u \cdot \frac{\mathring{h}_2}{\mathring{h}_1 + \mathring{h}_2} \right) \tag{8.4}$$

Der Koeffizient $\frac{\mathring{h}_1}{\mathring{h}_1 + \mathring{h}_2}$ gibt an, welchen Anteil der Radius $\mathring{h}_1$ an der Summe der beiden Radien $\mathring{h}_1$ und $\mathring{h}_2$ hat. Genau dieser Anteil von u wird von $\mathring{h}_1$ abgezogen. Dadurch wird oben beschriebene Fairness erreicht. Die resultierenden Hyperflächen erfüllen die Eigenschaft,

$$\mathring{h}_1 + \mathring{h}_2 \leq \delta(\vec{h}_1 + \vec{h}_2) \tag{8.5}$$

woraus die Überlappungsfreiheit folgt.

8.2 Programmablauf

Grundlage für die Realisierung ist der in Abschnitt 3.1 vorgestellte Algorithmus $\mathcal{L}$, dessen Resultat eine Menge H von Hyperflächen ist. Für die Durchführung sind neben den Trainingsdaten auch die Parameter r_{min} und r_{max} notwendig. Die Trainingsvektoren können auf Grund einer ungünstigen Reihenfolge eingeschlossene Hyperflächen produzieren. Eine Funktion $\mathcal{S}$ sortiert die Hyperflächen der Größe nach absteigend und gibt eine dieser Ordnung entsprechende Menge von Trainingsvektoren T' zurück. Mit dieser veränderten Reihenfolge wird $\mathcal{L}$ erneut ausgeführt, wodurch die Menge H' von Hyperflächen entsteht. Um eventuell auftretende Überlappungen zu beseitigen, wird die Funktion $\mathcal{U}$ angewendet, deren Resultat die endgültige Merkmalsraumaufteilung

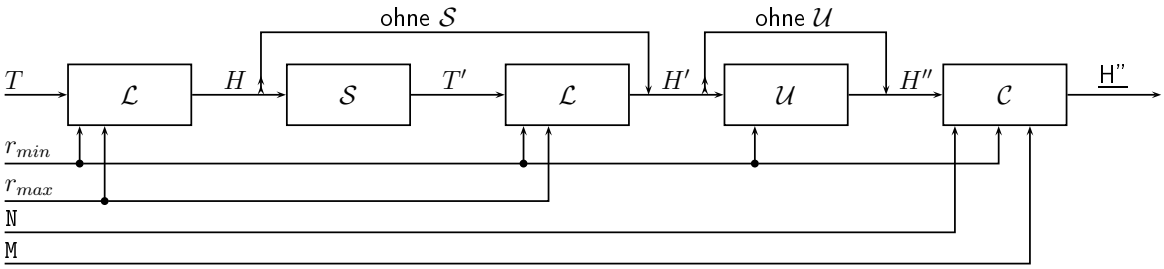


Abbildung 8.5: Programmablauf

H'' ist. Auch hier wird wieder r_{min} benötigt, da die Reduktion von eventuellen Überlappungen wieder r_{min} als untere Grenze hat. Anschließend wird H'' entsprechend den Anforderungen an die Konfigurationsleitung `r` codiert (siehe S. 33ff).

Für die Codierung sind wiederum die Parameter N, D und M erforderlich, die den Aufbau des Datenworts bestimmen. D lässt sich aus der Dimension der Trainingsdaten direkt ableiten, wogegen N (Bitbreite) und M (Anzahl der möglichen Hyperflächen) durch den Nutzer angegeben werden müssen. Entweder sind sie durch eine bestehende Synthetisierung vorgegeben oder können nach eigenem Ermessen gewählt werden. Die Belegung dieser Parameter kann unter Umständen ungeeignet sein, z.B. wenn die Anzahl der resultierenden Hyperflächen größer als M bzw. die angegebene Bitbreite nicht ausreicht, um die einzelnen Werte darzustellen. In diesem Fall wird eine Korrektur der Parameter nach oben durchgeführt. Es ist zu beachten, dass die dadurch entstehende Codierung mit dem Hardware-Layout nicht kompatibel ist, und die Synthese mit den neuen Werten wiederholt werden muss. Alternativ kann durch Modifikation von r_{min}, r_{max} oder den Trainingsdaten versucht werden, das Trainingsresultat in den gewünschten Grenzen zu halten.

Die beiden Erweiterungen des Algorithmus $\mathcal{S}$ und $\mathcal{U}$ sind optional und können bei Bedarf genutzt werden. Dadurch kann der Algorithmus genau, wie in [Nes95] vorgegeben, ausgeführt werden. Für eine genaue Beschreibung zur Schnittstelle des Programms sei auf Anhang D verwiesen.

9

Zusammenfassung und Ausblick

Die Klassifikation mit RCE-Netzwerken bietet eine leistungsfähige Alternative zu klassischen Verfahren. Vor allem die Möglichkeit, Klassen mit mehreren Häufungspunkten und unregelmäßigen Klassenstrukturen beschreiben zu können, hebt sich deutlich von anderen Verfahren (mit Ausnahme des Bayes-Klassifikators) ab. Bei Wahl einer geeigneten Metrik, wie Mahattan- oder Maximumsnorm, entsteht ein leistungsfähiger Klassifikator, der ohne Multiplikationen auskommt und daher gut in Hardware implementiert werden kann.

Echtzeitsignalverarbeitung erfordert hohen Datendurchsatz. Vor allem weil der Prozess der Klassifikation so weit wie möglich parallelisiert und nach dem Pipelining strukturiert wurde, ist es möglich, in jedem Takt eine Klassifikation durchzuführen. Der Platzbedarf ist demnach von der Anzahl der Hyperflächen und der Dimension des Merkmalsraums abhängig. In Kapitel 4 wurden weitere Möglichkeiten vorgestellt, den Klassifikator zu realisieren, die im Platzverbrauch weniger komplex waren, allerdings nicht mit maximalem Datendurchsatz arbeiten.

Die Synthese des Layouts kann über Parameter beeinflusst werden. Die Dimension des Merkmalsraums, die Auflösung der Merkmale und die Anzahl der Hyperflächen können separat eingestellt werden. Damit lässt sich nicht nur der Klassifikator optimal an die Klassifizierungsaufgabe anpassen, sondern auch der Platzverbrauch regulieren.

Wird der Klassifikator entsprechend der für die Feuererkennung im Kleinstsatelliten BIRD gewählten Parameter synthetisiert, so entsteht ein Layout, das selbst auf dem Low-Cost-FPGA Cyclone II nur 6.5% der verfügbaren Ressourcen benötigt.

Die Feuererkennung des BIRD war eine der wesentlichen Motivationen für diese Arbeit. Die erreichten Ergebnisse lassen den Schluss zu, dass der gewählte Entwurf geeignet ist, die Feuererkennung des BIRD durch FPGAs realisieren zu lassen, ohne die Bitbreite auf 5 Bit zu beschränken. Somit kann in Zukunft auf die Verwendung teurer Spezialhardware, wie den NI1000, verzichtet werden.

Der Trainingsalgorithmus konnte erfolgreich als Software implementiert werden. Neben der reinen Abbildung des durch [Nes95] beschriebenen Algorithmus wurde das Programm um zwei zusätzliche Optionen erweitert, die bei Bedarf genutzt werden können.

Da bei einer Realisierung in Hardware die Ressourcen stets eine wichtige Rolle spielen, kann mit der Erweiterung zur Beseitigung eingeschlossener Hyperflächen die Anzahl der Hyperflächen minimiert werden. Alle Hyperflächen, die ihren Mittelpunkt in einer Hyperfläche der eigenen Klassen haben, treten dann nicht mehr auf.

Darüber hinaus verfügt der Algorithmus über die Schwachstelle, Überlappungen fremdklassiger Hyperflächen zuzulassen, obwohl die Klassen gut separiert nebeneinander existieren könnten. Das liegt vor allem daran, dass in der ursprünglichen Algorithmik nur Trainingsvektoren mit Hyperflächen verglichen werden. Ein Vergleich der Hyperflächen untereinander findet nicht statt. Die zweite optionale Erweiterung widmet sich genau diesem Punkt. Nach Abschluss der Trainingsphase werden die Hyperflächen untereinander verglichen und evtl. auftretende Hyperflächen werden soweit verkleinert, bis sie keine Schnittmenge mehr haben.

Ausblick

Diese Arbeit ist ein erster Versuch den RCE-Netzwerk-Klassifikator in konfigurierbarer Hardware zu realisieren. Die erreichten Resultate sind für den Einsatz unter Echtzeitbedingungen geeignet und ermöglichen Durchsatzraten, die weit über den Möglichkeiten des NI1000 liegen. Diese Leistung fällt zu Lasten des Platzbedarfs, der mit der Dimension des Merkmalsraums und der Anzahl der Hyperflächen linear wächst. Für weiterführende Projekte wäre es denkbar, Lösungen zu entwerfen, die weniger komplex im Platzverbrauch sind und dadurch komplexe Klassifizierungsaufgaben ermöglichen.

Das Ziel der Echtzeitverarbeitung spielt nicht in allen Fällen eine wichtige Rolle. Fingerprint- und Texterkennungssysteme haben diese Anforderung nicht zwingend. Daher sollten für ein vielseitiges Anwendungsfeld die Möglichkeiten des Klassifikators ausgebaut werden. In Anlehnung an die generischen Parameter `N`, `D` und `M` ist ein zusätzlicher Parameter denkbar, mit dem der Platzverbrauch zwischen konstant und linear eingestellt werden kann. Da sich der Platzbedarf relativ genau berechnen lässt,

wäre sogar ein Automatismus denkbar, der für eine gegebene Hardware den besten Kompromiss zwischen paralleler und serieller Verarbeitung findet.

Eine andere Möglichkeit der Leistungssteigerung zeigt das folgende Beispiel: BIRD überfliegt weite Teile des Globus und arbeitet dabei auf beiden Hemisphären. Die atmosphärische Zusammensetzung beider Hemisphären ist dabei so unterschiedlich, dass die Merkmalsraumaufteilung unter der gleichen Problemstellung sehr unterschiedlich ausfällt. Werden beide Hemisphären in einer Merkmalsraumaufteilung beschrieben, so ist mit komplexen Klassenstrukturen zu rechnen. Betrachtet man hingegen die Merkmalsraumaufteilungen für beide Hemisphären separat, so ist deren Struktur bei weitem nicht so komplex. Wird nun die Hemisphäre *hem* als Merkmal in die Musterkennung aufgenommen, lässt sich zunächst zeigen, dass die Streuung aller Testdaten einer Klasse für alle *hem* sehr groß ausfällt. Betrachtet man dagegen alle Testdaten einer Klasse für ein bestimmtes *hem*, so ist die Streuung viel kleiner. Daraus folgt, dass die Komplexität der Merkmalsraumaufteilung für jedes *hem* viel geringer ist als für alle *hem* auf einmal. Der VHDL-Klassifikator ist so geschaffen, dass er jederzeit neu konfiguriert werden kann.

Zeigt sich nun, dass die Aufteilung des Merkmalsraums so stark von einer Variable abhängig ist, könnte in einem vorgesetzten einfachen Klassifikator die Konfiguration in Abhängigkeit dieser Größe ausgetauscht werden. Dann wird auf der nördlichen Hemisphäre eine Konfiguration genutzt, und auf der südlichen eine andere. Dadurch können die generischen Parameter `D` und `M` kleiner gewählt werden, wodurch das Layout weniger Platz benötigt. In einem Speicher müssen dann nur die unterschiedlichen Konfigurationen vorgehalten werden.

Der NI1000 realisiert neben den RCE-Netzwerken auch PRCE-Netzwerke (Probabilistic RCE), bei denen jeder Hyperfläche eine Wahrscheinlichkeitsdichtefunktion zu Grunde liegt. Die Klassenzugehörigkeit wird hier über die Wahrscheinlichkeit bestimmt. Diese Variante liefert auch bei Überlappungen eine eindeutige Zuordnung. Eine Erweiterung des VHDL-Entwurfs um dieses Verfahren würde sein Anwendungsfeld erneut steigern.

Denkbar wäre die Entwicklung einer umfassenden Mustererkennungs-Toolbox für die Arbeit auf konfigurierbarer Hardware, die Module für die Signalvorverarbeitung, Merkmalsextraktion und verschiedene Klassifikatoren enthält. Da Mustererkennung heutzutage in vielen Bereichen Anwendung findet und FPGAs hervorragend geeignet sind, sich entwickelnde Strukturen zu realisieren, zeigt sich hier ein interessantes und vielfältiges Beschäftigungsfeld auf.

Wie sich zeigt, bieten die RCE- und PRCE-Netzwerke noch viel Potential für weitere Entwicklungen. Da diese Klassifikatoren nach Betrachtung in Abschnitt 2.2.3 ihre Überlegenheit gegenüber anderen Verfahren beweisen konnten, sollte deren Leistungsfähigkeit bei der Bewältigung zukünftiger Klassifikationsaufgaben stets berücksichtigt werden.

A

Liste der Symbole

Symbol	Erklärung
$\vec{x}$	Vektor
x_i	Komponente eines Vektors
ω_i	eine Klasse
$\bar{x}$	Mittelpunkt
$\mathring{x}$	Radius
$\tilde{h}_i$	Bezeichnung der Klasse, zu dessen Beschreibung die Hyperfläche h_i dient.
Ω	Menge aller Klassen
H_i	Menge von Hyperflächen die die Klasse ω_i beschreiben
$\vec{x} \in \omega_i$	Muster $\vec{x}$ ist Exemplar der Klasse ω_i
M	Anzahl der Klassen
D	Dimension
$\mathcal{O}_{area}$	Platzbedarf
$\mathcal{O}_{time}$	Schaltungstiefe
$Latenz(g)$	Latenz der Komponente g
$\underline{a}$	binäre Darstellung von a
$\Theta(b)$	Anzahl der Bits von b
c'	Wort auf Leitung c
$a, b, c, ...$	konkrete Elemente
$i, j, k, ...$	Variable

B

Schnittstelle der Software

Die Trainingssoftware wurde mit IDL 6.1 für Windows implementiert. Sie erstellt aus einer Datei *lerningDataFile* mit den Trainingsdaten die Konfiguration des Klassifikators. Das Resultat wird in der Datei *configFile* gespeichert. Programmaufruf:

`RCE_Training,` *ri*, *ro*, *lerningDataFile*, *configFile*, *rmin*, *rmax*, *width*, *m*

Parameter	Typ	**Beschreibung**
ri	{0,1}	Flag, das angibt, ob eingeschlossene Hyperflächen beseitigt werden sollen. Wenn $ri = 1$, werden eingeschlossene Hyperflächen beseitigt, andernfalls nicht.
ro	{0,1}	Flag, das angibt, ob die Reduktion von Überlappungen ausgeführt werden soll. Wenn $ro = 1$, werden Überlappungen beseitigt, andernfalls nicht.
lerningDataFile	String	Pfad und Name der Datei, die die Trainingsdaten enthält
configFile	String	Pfad und Name der Datei, in der das Ergebnis gespeichert werden soll.
rmin	pos. Int	Mindestgröße r_{min} einer Hyperfläche
rmin	pos. Int	Maximale Größe r_{max} einer Hyperfläche
width	pos. Int	Bitbreite, mit der die einzelnen Werte aufgelöst werden sollen
m	pos. Int	Anzahl der Hyperflächen

lerningDataFile

Die Datei mit den Trainingsdaten muss im ASCII-Format vorliegen und ein Array von dezimal-codierten Integerwerten enthalten. Jede Zeile enthält die Komponenten eines Vektoren und an letzter Stelle die Nummer seiner Klasse. Die einzelnen Werte werden durch Leerzeichen getrennt.

Beispiel:

```
10 10 1
30 30 2
15 15 2
12 10 1
```

Dargestellt sind die Trainingsvektoren $\begin{pmatrix} 10 \\ 10 \end{pmatrix}$ und $\begin{pmatrix} 12 \\ 10 \end{pmatrix}$ der Klasse ω_1 und die Vektoren $\begin{pmatrix} 30 \\ 30 \end{pmatrix}$ und $\begin{pmatrix} 10 \\ 20 \end{pmatrix}$ der Klasse ω_2.

configFile

Das *configFile* enthält nach Abschluss des Trainings die resultierende Merkmalsraumaufteilung, dargestellt durch einen Hexadezimal-String. Neben dieser Information werden auch die verwendeten Parameter `N`, `D` und `M` dargestellt. Wurden die Eingangsparameter ungeeignet gewählt und musste eine Korrektur erfolgen, wird eine Warnung ausgegeben, und Eingangs- und Ausgangsparameter werden angezeigt.

Beispiel

Aufgerufen wird der Algorithmus mit den Trainingsdaten aus vorigem Abschnitt:

```
RCE_Training, 0 , 0 ,'daten3.txt', 'config.txt', 1, 50, 8, 4
```

Weder Überlappungen noch eingeschlossene Hyperflächen sollen beseitigt werden. r_{min} beträgt 1 und r_{max} 50. Gewünscht ist eine Auflösung von 8 Bit und es sollen 4 Hyperflächen dargestellt werden.

Das Konfigurationsfile hat dann folgenden Inhalt:

```
parameter:
   bits(n):           8
   dimensions(d):     2
   no. refVectors(m): 4

configuration:
--------------
0101090A0A0209140A02251E1E00000000
```

Werden statt 8 Bit nur 4 Bit zur Auflösung der Merkmale gewählt, kommt es zu beschriebener Anpassung der Bitbreite. Eine Wahl von $\texttt{M} = 3$ ergibt hier keine Probleme:

```
Warning!!! Parameter modified.

input parameter:
   Bits(n):           4
   Dimensions(d):     2
   No. RefVectors(m): 3

final parameter:
   bits(n):           8
   dimensions(d):     2
   no. refVectors(m): 3

configuration:
--------------
0101090A0A0209140A02251E1E
```

C

Das VHDL-Interface des Klassifikators

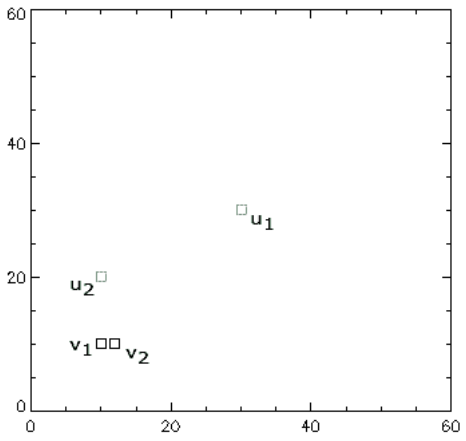
```
Library ieee;
Use ieee.std_logic_1164.all;

Library work;
Use work.tools.all;

Entity rce_classifier is

Generic(
      n: natural := 8; -- Bitbreite
      d: natural := 2; -- Dimension
      m: natural := 4   -- Anzahl der Hyperflächen
   );

Port(
      clock:   in std_logic;
      res:  in std_logic;
      enable: in std_logic;

      x:    in std_logic_vector((n*d)-1 downto 0);
      r:    in std_logic_vector((((m*(d+2))*n)-1)+n downto 0);
      class:   out std_logic_vector(n-1 downto 0);
      aff:  out AffType
   );

end rce_classifier;
```

D

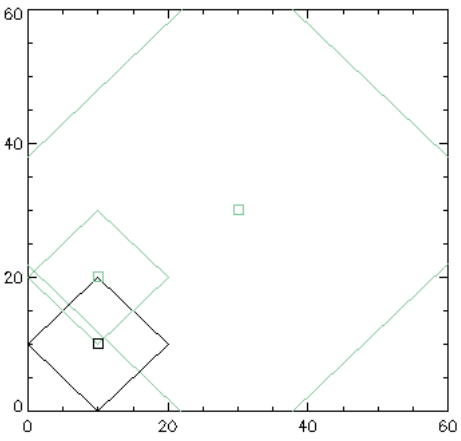
Ein Beispiel

Zu Grunde liegt der folgende Satz von Trainingsdaten:

$$\omega_1 = \left\{ \vec{v}_1 = \begin{pmatrix} 10 \\ 10 \end{pmatrix}; \vec{v}_2 = \begin{pmatrix} 12 \\ 10 \end{pmatrix} \right\}; \omega_2 = \left\{ \vec{u}_1 = \begin{pmatrix} 30 \\ 30 \end{pmatrix}; \vec{u}_2 = \begin{pmatrix} 10 \\ 20 \end{pmatrix} \right\}$$



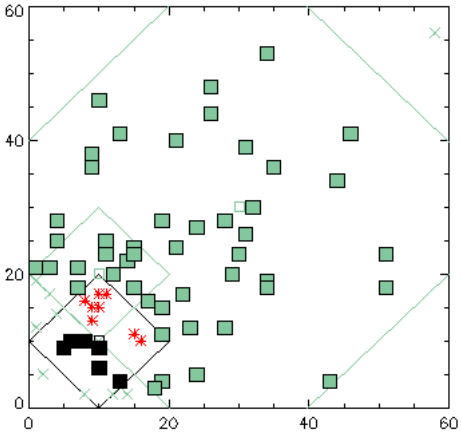
a: Die Punkte im Merkmalsraum



b: Resultierende Aufteilung

Sie wurden in der Reihenfolge $\vec{v}_1, \vec{u}_2, \vec{u}_1, \vec{v}_2$ durch den Trainingsalgorithmus verarbeitet. Daraus resultiert die Merkmalsraumaufteilung in Abbildung b. Zuerst wurde $\vec{v}_1$ als Hyperfläche mit Radius $r_{max} = 50$ aufgenommen. Nachdem im zweiten Schritt $\vec{u}_2$ in den Einflussbereich der Hyperfläche um $\vec{v}_1$ fällt, wird er zwar als Hyperfläche aufgenommen, jedoch werden die Radien beider Hyperflächen auf den Abstand der Vektoren gesetzt. Es folgt die Verarbeitung von $\vec{u}_1$, der zunächst als Hyperfläche mit

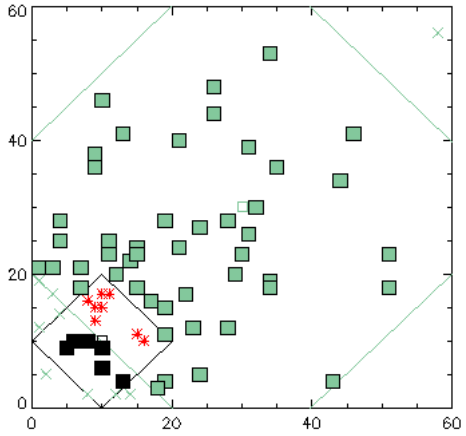
Radius $r_{max} = 50$ aufgenommen wird. In der zweiten Iteration des Algorithmus wird der Radius erst durch $\vec{v}_1$ zur Reduktion gezwungen und ein weiteres Mal durch $\vec{v}_2$. Wird $\vec{v}_2$ das erste Mal verarbeitet, wird keine neue Hyperfläche angelegt, da diese Region durch die Hyperfläche um $\vec{v}_1$ erfasst wird.



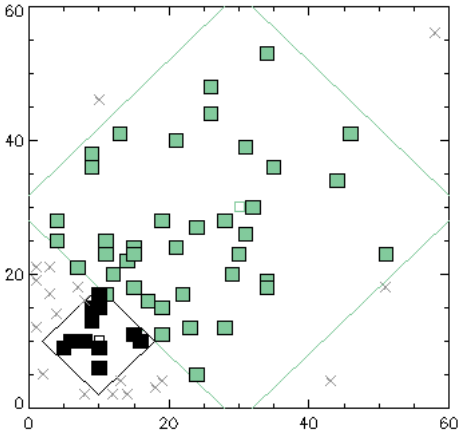
c: Klassifikation

Abbildung c zeigt nun ein Klassifizierungergebnis bei gewählter Merkmalsraumaufteilung. Neben korrekt klassifizierten Mustern (schwarze und grüne Vierecke) sind auch mehrfach zugeordnete Muster (rote Sterne) und nicht klassifizierte Muster (Kreuze) erkennbar.

Die Hyperfläche um $\vec{u}_2$ ist im Sinne von Abschnitt 8.1.1 eine eingeschlossene Hyperfläche. Durch Verwendung des in Abschnitt 8.1.1 vorgestellten Verfahrens können solche Hyperflächen vermieden werden. Abbildung d zeigt die Merkmalsraumaufteilung und Klassifiktation bei dieser Konfiguration.



d: ohne eing. Flächen



e: ohne Überlappungen

Die Aufteilung verfügt noch über eine Überlappung, die beseitigt werden kann, wenn die beiden Flächen so lange verkleinert werden, bis sie einander gerade noch berühren (Abbildung e).

Literaturverzeichnis

[Alt04] Altera.
Cyclone II - Device Family Data Sheet, 2004.

[Alt07] Altera.
Quartus II Development Software Handbook v7.1, 2007.

[DSH00] Richard O. Duda, David G. Stork, and Peter E. Hart.
Pattern Classification.
Wiley John + Sons, 2000.

[Hal99] Winfried Halle.
Ausgewählte Algorithmen der Segmentierung und Klassifikation zur thematischen Bildverarbeitung in der Fernerkundung.
Deutsches Zentrum für Luft- und Raumfahrt (DLR), 1999.

[Hof07] Dirk W. Hoffmann.
Grundlagen der technischen Informatik.
Hanser Fachbuchverlag, 2007.

[KB06] Frank Kesel and Ruben Bartholomä.
Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs.
Oldenbourg, 2006.

[LBB+04] E. Lorenz, W. Bärwald, K. Briess, H. Kayal, M. Schneller, and H. Wüsten.
Resumes Of The Bird Mission.
La Rochelle, 2004.

[MH04] Beate Meffert and Olaf Hochmuth.
Werkzeuge der Signalverarbeitung.
Pearson Studium, 2004.

[Nes95] Nest Inc.
NI1000 - Custom Developers' Guide, 1995.

[Sac06] Michael Sachs.
Wahrscheinlichkeitsrechnung und Statistik.
Hanser Fachbuchverlag, 2006.

[SH99] Samuel D. Sterns and Don R. Hush.
Digitale Verarbeitung analoger Signale, 7. Auflage.
Oldenbourg Verlag, 1999.

[Wan98] Markus Wannemacher.
Das FPGA-Kochbuch.
MITP, 1998.

Index