Informatik Praktikum
Erstellung eines Programms zur Testmustergenerierung

Markus Brehm
Betreuer: Felix Schürmann

Oktober 2001

Zusammenfassung

Im Rahmen des Informatikpraktikums sollte ein Programm zur Testmustergenerierung und -bearbeitung erstellt werden, das dann in der "electronic vision(s)" Gruppe am Kirchhoff-Institut für Physik eingesetzt werden soll.
In der "electronic vision(s)" Gruppe wurde ein Chip entwickelt, der ein neuronales Netzwerk auf Hardware-Basis implementiert. Um sinnvolle Aufgaben zu erfüllen, muß dieser Chip trainiert werden, d.h. ihm werden Testmuster mit entsprechenden Lösungen vorgelegt, die dann gelernt werden sollen.
Das im Rahmen dieses Praktikums entwickelte Programm soll die Generierung und Verwaltung von solchen Mustern vereinfachen. Entwicklungsziele waren vor allem eine große Allgemeinheit und Erweiterbarkeit. Das Programm wurde vollständig in C++ geschrieben, wobei als graphische Klassenbibliothek Qt von Trolltech verwendet wurde. Das Programm wurde unter Linux entwickelt und getestet, es sollte aber auch unter Windows lauffähig sein.

1 Motivation

1.1 Forschungsgegenstand der "electronic vision(s)" Gruppe

Die "electronic vision(s)" Gruppe am Kirchhoff-Institut für Physik der Universität Heidelberg entwickelt Chips, die auf Hardware-Ebene ein Neuronales Netz implementieren. Die Idee künstlicher neuronaler Netze gibt es schon seit längerem. Bisher werden diese allerdings hauptsächlich in Software realisiert. Dies bringt natürlich den Vorteil einer hohen Flexibilität, hat aber den Nachteil recht langsam zu sein. Hier setzt das Interesse der "electronic vision(s)" Gruppe an: Speziell entworfene Chips bringen einen erhebliche Geschwindigkeitvorteil. Außerdem kommen bei Benutzung analoger Schaltungen auch noch andere Effekte wie z.B. Rauschen ins Spiel, die auch in den als Vorbild dienenden natürlichen neuronalen Netzen vorhanden sind, die aber in reinen Software-Lösungen entweder fehlen, oder nur sehr schwer und unvollständig zu realisieren sind.

1.2 Bisheriges Vorgehen

Bisher wurden Testmuster durch speziell geschriebene C Programme oder per Hand erzeugt und dann abgespeichert. Dieses Verfahren war in mehrerer Hinsicht nicht ganz zufriedenstellend:

- Es mußte für jeden Typ von Testmuster ein eigenes Programm erstellt werden.
- Eine Auswahl, ob ein Muster zu Test- oder Trainingszwecken dienen sollte, war nicht möglich
- Eine Änderung von Mustern oder deren Gewichtung war nur durch manuelles Editieren einer Textdatei möglich
- Zeitlich veränderbare Muster waren nicht vorgesehen
- Eine Anwendung von Operatoren wie z.B. Rauschen, Verschieben usw. war nicht vorgesehen
- Eine Verwaltung von Mustern (kopieren, löschen) war nicht möglich
- Muster konnten nur binär sein, Muster die aus ganzen Zahlen bestehen waren nicht möglich

2 Entwurf

Die Schwächen des bisherigen Vorgehens gaben die Anforderung an das neue Programm vor. Die neuen Anforderungen machten zwei Dinge notwendig: Die Schaffung einer neuen Datenstruktur, die insbesondere auch mit zeitlichen Mustern umzugehen weiß und das Erstellen einer graphischen Benutzeroberfläche.

2.1 Die neue Datenstruktur

Grundelement der neuen Struktur ist ein Pattern (Muster). Patterns besitzen ein Ein- und ein Ausgabemuster und kennen ihr Gewicht. Ein Pattern ist immer ein zeitlich konstantes Muster. Ein sich zeitlich änderndes Muster wird dadurch realisiert, daß mehrere Patterns zusammengefaßt werden.
Dies geschieht dann in einem PatternFrame. Außer der Zusammenfassung von Mustern zu verschiedenen Zeiten faßt ein Frame auch verschiedene Muster zu einer Einheit zusammen. Ein Frame zeichnet sich dabei dadurch aus, daß alle seine Mitglieder-Muster gleiche Zeit-Tiefe und gleiche Anzahl an Ein- und Ausgabeneuronen besitzen.
Mehrere Patternframes werden schließlich durch einen PatternHandler verwaltet.

2.2 Die graphische Benutzeroberfläche

Anforderungen an die graphische Benutzeroberfläche waren:

- Einfache Navigation durch Frames und Patterns
- Übersichtliche Darstellung möglichst vieler Muster

- Bereitstellung von verschieden Werkzeugen zur Erstellung und Bearbeitung von Mustern
- Gute Erweiterbarkeit, besonders im Hinblick auf die bereitgestellten Werkzeuge

Diese Forderungen legten eine Dreiteilung des Bildschirms nahe (Vgl. Screenshot Abb.1):

- Ein Bereich zum Navigieren in Form eines Baumes
- Ein Bereich in Form einer Tabelle zur Darstellung eines Teils der Muster
- Ein Bereich in Form eines Karteikartenreiters, in dem sich die Werkzeuge befinden

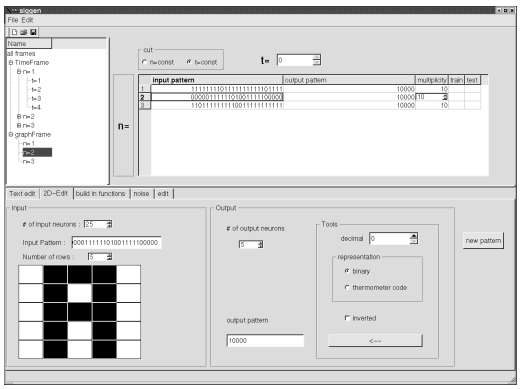

Abbildung 1: Screenshot

3 Implementierung der neuen Datenstruktur

Die Implementierung der neuen Datenstruktur war nicht teil dieses Praktikums. Ihre Verwendung ist auch nicht auf die Erzeugung von Mustern beschränkt, so wird sie auch zur Umsetzung der Testmuster in elektrische Signale für den Chip eingesetzt werden.
Da sie jedoch parallel zu diesem Praktikumsprojekt entwickelt wurde und das Projekt auf ihr basiert, möchte ich doch einige Worte darüber verlieren.

3.1 Die Klasse Pattern

Die Klasse Pattern repräsentiert ein zeitlich konstantes Muster. Dieses Muster besteht aus einem Eingabe- und einem Ausgabemuster, sowie einem Gewicht. Bei dem Ein- bzw. Ausgabemuster kann es sich um ein binäres Muster oder um ein Muster bestehend aus ganzen Zahlen (integer) handeln. Intern wird jedes Muster als Integer-Muster gespeichert.
Wichtige Funktionen der Klasse Pattern sind Funktionen zur Abfrage und zum Ändern von Ein- und Ausgabemustern sowie des Gewichts.

3.2 Die Klasse PatternFrame

Die Klasse PatternFrame ist eine abstrakte Klasse, das heißt ihre sämtlichen Funktionen sind als virtual deklariert und enthalten keine Anweisungen. Es handelt sich bei PatternFrame also um eine Interface-Klasse, also eine Klasse die nur die Schnittstellen festlegt. Ein PatternFrame verwaltet n*t Patterns (n: Anzahl der Muster, t:Zeittiefe) (Vgl. Abb. 2)
Wichtige Funktionen, die PatternFrame bereitstellt, ist die Rückgabe und das Setzen eines Patterns, die Erhöhung und Verminderung der Dimensionen des Frames (Anzahl der Muster, Zeittiefe, Anzahl der Ein- und Ausgabeneuronen), das Vertauschen von Patterns usw. Außerdem besitzt ein PatternFrame einen Namen, der gesetzt und abgefragt werden kann.
Zur Zeit gibt es eine Implementierung des PatternFrames, das VAPatternFrame; es speichert die Patterns in einem Valarray.

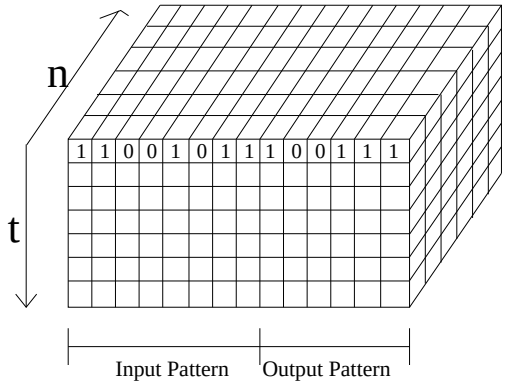

Abbildung 2: Ein Frame faßt n*t Patterns zusammen

3.3 Die Klasse PatternHandler

Auch die Klasse PatternHandler, die die Verwaltung der Frames übernimmt, ist eine Interface Klasse.
Wichtige Funktionen sind das Laden und Speichern von Dateien, der Zugriff auf einzelne Frames, sowie das Einfügen und Löschen von Frames. Außerdem ist vorgesehen, daß ein PatternHandler einen Frame expandiern kann, das heißt, gewünschte Operationen, wie z.B. Rauschen, auf ein Frame anwenden kann.
Zur Zeit existiert ein SimplePatternHandler, der die meisten Basisfunktionen implementiert und zur Speicherung in einer Datei noch das alte Format verwendet. Ein XMLPatternHandler, der dann alle Funktionen und das neue XML-Format unterstützt wird zur Zeit entwickelt.

4 Implementierung der graphischen Benutzeroberfläche

4.1 Überblick

Die Dreiteilung des Bildschirms spiegelt sich auch in der Klassenstruktur wider. Die Klasse MainWin stellt das Hauptfenster dar, das sich aus den drei Fenstern entsprechend den Klassen FrameTreeView, OverViewFormImpl und MyTabwidget zusammensetzt. Vergleiche hierzu auch Abb. 3. Zu beachten ist, daß es sich bei dieser Abbildung nicht um ein klassisches Klassendiagramm (also eine Darstellung der Vererbungsstruktur), sondern um ein Diagramm handelt, das die Benutzung der Klassen darstellt.
Im folgenden soll kurz erläuteret werden, welche Aufgaben die jeweiligen Klassen erfüllen.

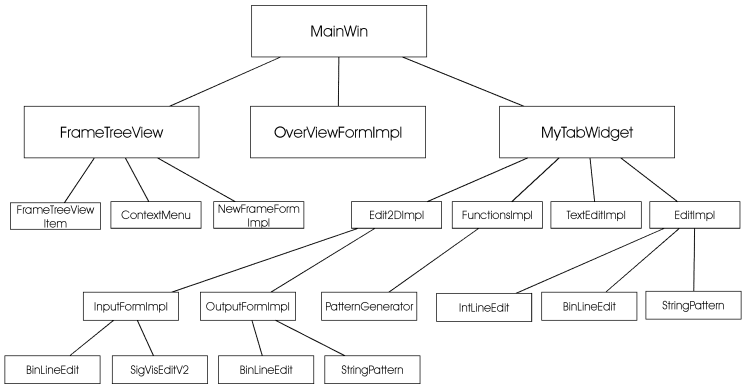

Abbildung 3: Beutzung der Klassen:
Klassen, die weiter oberhalb im Diagramm stehen benutzen die darunterliegenden Klassen, wenn sie mit einer Linie verbunden sind.

4.2 Die Klasse MainWin

Die Klasse MainWin repräsentiert das Hauptfenster und bindet die drei Klassen FrameTreeView, OverViewFormImpl und MyTabwidget ein. Außerdem ist sie für die Darstellung der Menü- sowie der Statuszeile verantwortlich und übernimmt Funktionen, die alle anderen Programmteile betreffen, wie das Laden und Speichern von Dateien.

4.3 Die Klasse OverViewFormImpl

Die Klasse OverViewImpl dient der Darstellung eines Schnitts des aktuell ausgewählten Frames in einer Tabelle. Sie beinhaltet ebenfalls Funktionen zur Änderung der Art des Schnittes (t=const oder n=const) sowie der Position des Schnitts (t bzw. n).
Wurde ein neues Muster ausgewählt, wird das Signal active_pattern_changed gesendet.

4.4 Die Klasse FrameTreeView

FrameTreeView übernimmt die Darstellung der Frames in einer Baumstruktur, wobei sie sich der Klasse FrameTreeViewItem bedient. FrameTreeView leitet sich dabei von QListView ab.
Außerdem kann sie ein Kontext-Menü einblenden, wofür sie sich der Klasse ContextMenu bedient. Um die in diesem Kontext-Menü aufgeführten Befehle ausführen zu können, besitzt sie Funktionen zum Löschen, Kopieren und Einfügen von Frames und Mustern.
Wenn ein neues Muster ausgewählt wurde, sendet die Klasse das Signal new_selected, wurden Änderungen an Frames vorgenommen, das Signal pleaseReread.

4.5 Die Klasse FrameTreeViewItem

Die Klasse FrameTreeViewItem ist von QListViewItem abgeleitet und stellt die von FrameTreeView benutzten Elemente zur Darstellung der Framestruktur da. In Ergänzung der Methoden von QListViewItem besitz sie Funktionen, die das repräsentierte Frame, sowie die repräsentierte n und t Koordinate widergeben.

4.6 Die Klasse ContextMenu

Das Einzige, worin sich die Klasse ContextMenu von ihrem Vater QPopUpMenu unterscheidet, ist, daß ContextMenu geschlossen wird, sobald sich der Mauszeiger aus ihrem Berich herausbewegt. Context Menu wird von FrameTreeView verwendet, sobald die rechte Maustaste gedrückt wurde.

4.7 Die Klasse NewFrameFormImpl

Die Klasse NewFrameFormImpl enthält den Dialog zur Erzeugung eines neuen Frames und wird ebenfalls von FrameTreeView verwendet.

4.8 Die Klasse MyTabWidget

Die Klasse MyTabWidget dient der Aufnahme verschiedener Werkzeuge in Form von Karteikarten-Reitern und ist deshalb von QTabWidget abgeleitet. Sie reagiert auf das Signal new_selected und leitet es an die Werkzeuge, die darauf reagieren müssen weiter. Außerdem empfängt sie von den Werkzeugen das Signal pleaseReread und leitet es an FrameTreeView weiter.
Zur Zeit benutzt MyTabWidget die Klassen Edit2DImpl, FunctionsImpl,TextEditImpl, NoiseImpl und EditImpl. Es ist vorgesehen, daß an dieser Stelle weitere Werkzeuge eingefügt werden.

4.9 Die Klasse TextEditImpl

TextEditImpl dient zum manuellen Editieren der Dateien, in denen die Muster gespeichert werden. Diese Funktionalität ist zur Zeit noch nicht implementiert.

4.10 Die Klasse NoiseImpl

Hier ist vorgesehenen Operatoren, wie z.B. Rauschen, auf Frames anzuwenden. Auch diese Funktion ist noch nicht implementiert.

4.11 Die Klasse FunctionsImpl

Diese Klasse dient der Erstellung von Frames mit zugehörigen Muster aus Funktionen. Zur Zeit ist nur das Erstellen eines n-bit-Parity Musters möglich. FunctionsImpl fragt hierzu in einem Dialog die entsprechenden Parameter ab und ruft dann die Klasse PatternGenerator auf, die das Frame dann tatsächlich erzeugt.

4.12 Die Klasse PatternGenerator

PatternGenerator gibt ein Frame zurück. Zur Zeit besitzt PatternGenerator nur eine Methode generateParity zur Erzeugung eines n-bit-Parity Musters. Weitere Methoden können leicht hinzugefügt werden.

4.13 Die Klasse EditImpl

Die Klasse EditImpl erlaubt das manuelle Editieren einzelner Muster, wozu sie die Klassen BinLineEdit (für binäre Muster) bzw. IntLineEdit (für integer Muster) benutzt. Als Hilfsmittel stellt sie eine Konvertierung von im Dezimalsystem codierten Zahlen in das Binärsystem bzw. in das Thermometercodesystem zur Verfügung, wozu sie auf die Klasse StringPattern zurückgreift.

4.14 Die Klasse BinLineEdit

Die Klasse BinLineEdit dient dem Editieren binärer Muster. Die Länge der Muster kann mit der Funktion setMyMaxLength gesetzt werden. Bei einer Änderung des Musters sendet BinLineEdit das Signal PatternStringChanged, bei einer Änderung der Cursor-Position das Signal newCursorPosition (zur Anzeige in der Statuszeile).

4.15 Die Klasse IntLineEdit

Die Klasse IntLineEdit dient dem Editiern von Integer-Mustern. Ansonsten gilt das für BinLineEdit Gesagte.

4.16 Die Klasse StringPattern

Die Klasse StringPattern dient dem Konvertieren von Dezimalzahlen in das Binär- bzw. in das Thermometercodesystem. Hierzu kann mittels setFromInt ein Muster erzeugt und mittels getString der entsprechende String abgefragt werden.

4.17 Die Klasse Edit2DImpl

Die Aufgabe der Klasse Edit2DImpl besteht in der graphischen Veränderung eines Eingabemusters und dem nicht-graphischem Editieren des Ausgabemusters. Für die erste Aufgabe bedient sie sich der Klasse InputFormImpl, für die zweite der Klasse OutputFormImpl.

4.18 Die Klasse OutputFormImpl

Die Klasse OutputFormImpl hat ähnliche Aufgaben wie EditImpl und funktioniert deshalb auch analog zu dieser. Allerdings beschränkt sie sich auf das Ausgabemuster.

4.19 Die Klasse InputFormImpl

Die Klasse InputFormImpl bietet die Möglichkeit, das (binäre) Eingabemuster graphisch zu verändern. Dazu benutzt sie die Klasse SigVisEditV2. Außerdem erlaubt sie auch das direkte Bearbeiten des Eingabemusters, wozu sie die Klasse BinLinEdit benutzt. Des weiteren sorgt sie dafür, daß beide Anzeigen (graphische und nicht-graphische) stets übereinstimmen.
InputFormImpl erlaubt auch die Parameter Anzahl der Eingabeneuronen und Anzahl der Zeilen, in denen das Eingabemuster dargestellt werden soll, zu ändern.

4.20 Die Klasse SigVisEditV2

Die Klasse SigVisEditV2 stellt ein rechteckiges Widget zur graphischen Veränderung eines binären Musters zur Verfügung. Die Parameter wie Anzahl der Eingabeneuronen und Anzahl der Zeilen, in denen das Eingabemuster dargestellt werden soll, werden im Konstruktor übergeben. Es ist deshalb notwendig, bei einer Änderung eines dieser Paramter ein neues SigVisEditV2 zu erzeugen.
Wird das Muster verändert, so sendet die Klasse das Signal patternChanged. Außerdem sendet sie die Koordinate des Mauszeigers via newStatusMessage zur Anzeige in der Statusleiste.

4.21 Beispiel für den Signalfluß

Nachdem in den vorangegangenen Abschnitten die einzelnen Klassen beschrieben wurden, soll in diesem Abschnitt versucht werden, wieder den Blick für das Ganze zurückzugewinnen. Um zu sehen, wie die einzelnen Klassen zusammenarbeiten, soll hier beschrieben werden, welche Aktionen durch das Auswählen eines neuen Musters in der Baumansicht veranlaßt werden.
Wird ein neues Element (Frame oder Muster) ausgewählt, so sendet FrameTreeView das Signal currentChanged, welches von new_selected_slot (auch FrameTreeView) aufgefangen wird. Dieser Slot stellt dann fest, welcher Frame und welche n- und t- Koordinaten ausgewählt sind und sendet das Signal new_selected, das diese Angaben enthält.
Dieses Signal wird dann von display_new_slot der Klassen OverViewFormImpl sowie MyTabWidget aufgefangen. In OverViewFormImpl wird im Bedarfsfall die Tabelle neu gezeichnet und die Markierung geändert. In MyTabWidget werden die Funktionen display_new der Klassen EditImpl sowie Edit2DImpl aufgerufen. EditImpl ebenso wie Edit2DImpl aktualisiert daraufhin das gerade bearbeitete Muster.

5 Erweiterungen

Aus Zeitgründen konnten im Rahmen des Informatik-Praktikums nicht alle wünschenswerten Funktionen implementiert werden. Insbesondere fehlt noch die vollständige Unterstützung von Integer-Mustern, sowie das Anwenden von Funktionen wie z.B. Rauschen auf Frames.

5.1 Notwendige Maßnahmen zur Implementierung der Unterstützung von Integer-Mustern

- In FrameTreeView muß eine Möglichkeit geschaffen werden, einen Frame als Integer- oder Binärframe zu kennzeichnen (z.B. beim Erstellen eines neuen Frames).
- In OverViewFormImpl muß vor dem Zeichnen der Tabelle festgestellt werden, ob es sich um ein Integer- oder Binärmuster handelt. Entsprechend muß die Ausgabe in die Tabelle in einem anderen Format erfolgen.
- In EditImpl muß, falls es sich um ein Integer-Muster handelt die Klasse IntLineEdit anstatt von BinLineEdit verwendet werden.

5.2 Notwendige Maßnahmen beim Hinzufügen weiterer Werkzeuge (wie z.B. einem Rauschoperator)

Ein neues Werkzeug wird einfach als neues Widget in MyTabWidget eingebunden (im Konstruktor). Verändert es Muster oder Frames, so muß es das Signal pleaseReread senden; dieses Signal muß im Konstruktor von MyTabWidget mit dessen Signal pleaseReread verbunden werden. Soll das neue Werkzeug auf das Auswählen eines neuen Musters reagieren, so muß die entsprechende Routine von display_new_slot (MyTabWidget) aus aufgerufen werden. Benötigt es einen PatternHandler, so muß dieser von setPatternHandler (MyTabWidget) aus gesetzt werden.

6 Zusammenfassung

Im Rahmen dieses Informatik-Praktikums wurde ein Programm zur Erzeugung und Bearbeitung von Testmustern erstellt, das in der "electronic vision(s)" Gruppe am Kirchhoff Institut für Physik der Universität Heidelberg eingesetzt werden soll. Dabei wurde eine Benutzeroberfläche entworfen und eine Basisfunktionalität implementiert. Weitere Funktionalität sollte mit vertretbarem Aufwand hinzuzufügen sein. Zu den zur Zeit vorhandenen Funktionen gehören:

- Das Verwalten von Frames und Mustern (Erzeugen,Kopieren,Löschen)
- Das Anzeigen von Frames und Mustern
- Das Bearbeiten von Mustern (auch zeitlich veränderlichen)
- Das graphische Bearbeiten von Mustern
- Das Erzeugen von n-bit-Parity Mustern